
Article:
Paterson, Colin orcid.org/0000-0002-6678-3752 and Calinescu, Radu Constantin orcid.org/0000-0002-2678-9260 (2020) Observation-Enhanced QoS Analysis of Component-Based Systems. IEEE Transactions on Software Engineering. pp. 526-548. ISSN 0098-5589

https://doi.org/10.1109/TSE.2018.2864159

Observation-Enhanced QoS Analysis of Component-Based Systems

Colin Paterson, Radu Calinescu

Abstract—We present a new method for the accurate analysis of the quality-of-service (QoS) properties of component-based systems. Our method takes as input a QoS property of interest and a high-level continuous-time Markov chain (CTMC) model of the analysed system, and refines this CTMC based on observations of the execution times of the system components. The refined CTMC can then be analysed with existing probabilistic model checkers to accurately predict the value of the QoS property. The paper describes the theoretical foundation underlying this model refinement, the tool we developed to automate it, and two case studies that apply our QoS analysis method to a service-based system implemented using public web services and to an IT support system at a large university, respectively. Our experiments show that traditional CTMC-based QoS analysis can produce highly inaccurate results and may lead to invalid engineering and business decisions. In contrast, our new method reduced QoS analysis errors by 84.4–89.6% for the service-based system and by 94.7–97% for the IT support system, significantly lowering the risk of such invalid decisions.

Index Terms—Quality of service, component-based systems, Markov models, probabilistic model checking.

1 Introduction

Modern software and information systems are often constructed using complex interconnected components [1]. The performance, cost, resource use and other quality-of-service (QoS) properties of these systems underpin important engineering and business decisions. As such, the QoS analysis of component-based systems has been the subject of intense research [2], [3], [4], [5]. The solutions devised by this research can analyse a broad range of QoS properties by using *performance models* such as Petri Nets [6], [7], layered queuing networks [8], Markov chains [9], [10] and timed automata [11], together with tools for their simulation (e.g. Palladio [12] and GreatSPN [13]) and formal verification (e.g. PRISM [14] and UPPAAL [15]).

These advances enable the effective analysis of many types of performance models. However, they cannot support the design and verification of real systems unless the analysed models are accurate representations of the system behaviour, and ensuring the accuracy of performance models remains a major challenge. Our paper address this challenge for *continuous-time Markov chains* (CTMCs), a type of stochastic state transition models used for QoS analysis at both design time [10], [16], [17] and runtime [18], [19]. To this end, we present a tool-supported method for Observation-based Markov chaiN refInement (OMNI) and accurate QoS analysis of component-based systems.

The OMNI method comprises the five activities shown in Fig. 1. The key characteristic of OMNI is its use of observed execution times for the components of the analysed system to refine a high-level abstract CTMC whose states correspond to the operations executed by these components. As such, the first OMNI activity is the collection of these execution time observations, which can come from unit testing the components prior to system integration, from logs of other systems that use the same components, or from the log of the analysed system. The second OMNI activity involves the development of a high-level CTMC model of the system under analysis. This model can be generated from more general software models such as annotated UML activity diagrams as in [16], [20], or can be provided by the system developers. The next OMNI activity requires the formalisation of the QoS properties of interest as continuous stochastic logic formulae.

The fourth activity of our OMNI method is the refinement of the high-level model. OMNI avoids the synthesis of unnecessarily large and inefficient-to-analyse models by generating a different refined CTMC for each QoS property of interest. This generation of property-specific CTMCs is fully automated and comprises two steps. The first step, called *component classification*, determines the effect of every system component on the analysed QoS property. The second step, called *selective refinement*, produces the property-specific CTMC by using phase-type distributions [21] to refine only those parts of the high-level CTMC that correspond to components which influence the QoS property being analysed. As such, OMNI-refined CTMCs model component executions with much greater accuracy than traditional CTMC modelling, whose exponential distributions match only the first moment of the unknown distributions of the observed execution times.

In the last activity of our method, the refined CTMC models generated by OMNI are analysed with the established probabilistic model checker PRISM [14]. As illustrated by the two case studies presented in the paper, these models support the accurate and efficient analysis of a broad spectrum of QoS properties specified in continuous stochastic logic [22]. As such, OMNI's observation-enhanced QoS analysis can prevent many invalid engineering and business decisions associated with traditional CTMC-based QoS analysis.

• *C. Paterson and R. Calinescu are with the Department of Computer Science at the University of York, UK.*

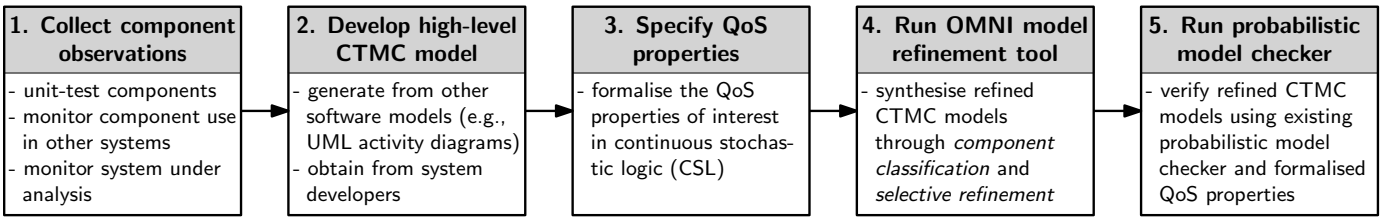


Fig. 1: OMNI workflow for the QoS analysis of component-based systems

The OMNI activities 2, 3 and 5 correspond to the traditional method for QoS property analysis through probabilistic model checking. Detailed descriptions of these activities are available (e.g., in [23], [24], [25], [26], [27]) and therefore we do not focus on them in this paper. Activities 1 and 4 are specific to OMNI. However, the tasks from activity 1 are standard software engineering practices, so the focus of our paper is on the observation-based refinement techniques used in the fourth activity of the OMNI workflow.

Like most methods for software performance engineering [28], [29], [30], OMNI supports both the design of new systems and the verification of existing systems. Using OMNI to assess whether a system under design meets its QoS requirements or to decide a feasible service-level agreement for a system being developed requires the collection of component observations by unit testing the intended system components, or by monitoring other systems that use these components. In contrast, for the verification of the QoS properties of an existing system, component observations can be collected using any of the techniques listed under the first activity from Fig. 1, or a combination thereof.

A preliminary version of OMNI that did not include the component classification step was introduced in [31]. This paper extends the theoretical foundation from [31] with key results that enable component classification, and therefore the synthesis of much smaller and faster to analyse refined CTMCs than those generated by our preliminary OMNI version. This extension is presented in Section 4.2, implemented by our new OMNI tool described in Section 5, and shown to reduce verification times by 54–74% (compared to the preliminary OMNI version) in Section 6.5. This is a particularly significant improvement because the same QoS property is often verified many times, to identify suitable values for the parameters of the modelled system (e.g., see the case studies from [32], [33], [34], [35], [36]). Additionally, we considerably extended and improved the validation of OMNI by evaluating it for the following two systems:

1) A service-based system that we implemented using six real-world web services — two commercial web services provided by Thales Group, three free Bing web services provided by Microsoft, and a free WebserviceX.Net web service. The evaluation of OMNI for this system was based on lab experiments.
2) The IT support system at the Federal Institute of Education, Science and Technology of Rio Grande do Norte (IFRN), Brazil. This system has over 44,000 users — students and IFRN employees (including IT support staff). The evaluation of OMNI for this system was based on real datasets obtained from the system logs.

The rest of the paper is structured as follows. Section 2 introduces the notation, terminology and theoretical background for our work. Section 3 describes the service-based system used to evaluate OMNI, as well as to motivate and illustrate our QoS analysis method throughout the paper. The assumptions and theoretical results underlying the component classification and selective refinement steps of OMNI are presented in Section 4, and the tool that automates their application is described in Section 5. Section 6 evaluates the effectiveness of OMNI for the two systems mentioned above. This evaluation shows that, compared to traditional CTMC-based QoS analysis, our method (a) reduces analysis errors by 84.4–89.6% for the service-based system and by 94.7–97% for the IT support system; and (b) lowers the risk of invalid engineering and business decisions. The experimental results also show a decrease of up to 71.4% in QoS analysis time compared to our preliminary OMNI version from [31]. Section 7 discusses the threats to the validity of our results. The paper concludes with an overview of related work in Section 8 and a brief summary in Section 9.

2 Preliminaries

2.1 Continuous-time Markov chains

Continuous-time Markov chains [37] are mathematical models for continuous-time stochastic processes over countable state spaces. To support the presentation of OMNI, we will use the following formal definition adapted from [32], [38].

Definition 2.1. A continuous-time Markov chain (CTMC) is a tuple

$$\mathcal{M} = (S, \boldsymbol{\pi}, \mathbf{R}), \tag{1}$$

where S is a finite set of states, $\boldsymbol{\pi} : S \to [0,1]$ is an initial-state probability vector such that the probability that the CTMC is initially in state $s_i \in S$ is given by $\boldsymbol{\pi}(s_i)$ and $\sum_{s_i \in S} \boldsymbol{\pi}(s_i) = 1$, and $\mathbf{R} : S \times S \to \mathbb{R}$ is a transition rate matrix such that, for any states $s_i \neq s_j$ from S, $\mathbf{R}(s_i, s_j) \geq 0$ specifies the rate with which the CTMC transitions from state s_i to state s_j, and $\mathbf{R}(s_i, s_i) = -\sum_{s_j \in S \setminus \{s_i\}} \mathbf{R}(s_i, s_j)$.

We will use the notation $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ for the continuous-time Markov chain $\mathcal{M}$ from (1). The probability that this CTMC will transition from state s_i to another state within t time units is

$$1 - e^{-t \cdot \sum_{s_k \in S \setminus \{s_i\}} \mathbf{R}(s_i, s_k)}$$

and the probability that the new state is $s_j \in S \setminus \{s_i\}$ is

$$p_{ij} = \mathbf{R}(s_i, s_j) \;/ \sum_{s_k \in S \setminus \{s_i\}} \mathbf{R}(s_i, s_k). \tag{2}$$

A state s is an *absorbing state* if $\mathbf{R}(s,s') = 0$ for all $s' \in S$, and a *transient state* otherwise.

The properties of a CTMC $\mathcal{M}$ are analysed over its set of finite and infinite paths $Paths^{\mathcal{M}}$. A *finite path* is a sequence $s_1 t_1 s_2 t_2 \ldots s_{k-1} t_{k-1} s_k$, where $s_1, s_2, \ldots, s_k \in S$, $\boldsymbol{\pi}(s_1) > 0$, s_k is an absorbing state, and, for all $i = 1, 2, \ldots, k-1$, $\mathbf{R}(s_i, s_{i+1}) > 0$ and $t_i > 0$ is the time spent in state s_i. An *infinite path* from $Paths^{\mathcal{M}}$ is an infinite sequence $s_1 t_1 s_2 t_2 \ldots$ where $\boldsymbol{\pi}(s_1) > 0$, and, for all $i \geq 1$, $s_i \in S$, $\mathrm{R}(s_i, s_{i+1}) > 0$ and the time spent in state s_i is $t_i > 0$. For any path $\omega \in Paths^{\mathcal{M}}$, the state occupied by the path at time $t \geq 0$ is denoted $\omega@t$. For infinite paths, $\omega@t = s_i$, where i is the smallest index for which $t \leq \sum_{j=1}^{i} t_j$. For finite paths, $\omega@t$ is defined similarly if $t \leq \sum_{j=1}^{k-1} t_j$, and $\omega@t = s_k$ otherwise. Finally, the i-th state on the path ω is denoted $\omega[i]$, where $i \in \mathbb{N}_{>0}$ for infinite paths and $i \in \{1, 2, \ldots, k\}$ for finite paths.

Continuous-time Markov chains are widely used for the modelling and analysis of stochastic systems and processes from domains as diverse as engineering, biology and economics [39], [40]. In this paper, we focus on the use of CTMCs for the modelling and QoS analysis of component-based software and IT systems. These systems are increasingly important for numerous practical applications, and advanced probabilistic model checkers such as PRISM [14], MRMC [41] and Storm [42] are available for the efficient analysis of their CTMC models.

2.2 Continuous stochastic logic

CTMCs support the analysis of QoS properties expressed in *continuous stochastic logic* (CSL) [22], which is a temporal logic with the syntax defined below.

Definition 2.2. Let AP be a set of atomic propositions, $a \in AP$, $p \in [0,1]$, I an interval in $\mathbb{R}$ and $\bowtie \in \{\geq, >, <, \leq\}$. Then a state formula Φ and a path formula Ψ in continuous stochastic logic are defined by the following grammar:

$$\begin{array}{l} \Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid P_{\bowtie p}[\Psi] \mid \mathcal{S}_{\bowtie p}[\Phi] \\ \Psi ::= X\Phi \mid \Phi U^I \Phi \end{array} . \quad (3)$$

CSL formulae are interpreted over a CTMC whose states are *labelled* with atomic propositions from AP by a function $L : S \to 2^{AP}$. The (*transient-state*) *probabilistic operator* P and the *steady-state operator* $\mathcal{S}$ define bounds on the probability of system evolution. *Next path formulae* $X\Phi$ and *until path formulae* $\Phi_1 U^I \Phi_2$ can occur only inside the probabilistic operator P.

The semantics of CSL is defined with a satisfaction relation $\models$ over the states $s \in S$ and the paths $\omega \in Paths^{\mathcal{M}}$ of a CTMC [38]. OMNI improves the analysis of QoS properties expressed in the transient fragment of CSL,[1] with semantics defined recursively by:

$$\begin{array}{ll} s \models true & \forall s \in S \\ s \models a & \text{iff } a \in L(s) \\ s \models \Phi_1 \wedge \Phi_2 & \text{iff } s \models \Phi_1 \wedge s \models \Phi_2 \\ s \models \neg\Phi & \text{iff } \neg(s \models \Phi) \\ s \models P_{\bowtie p}[\Psi] & \text{iff } Pr_s\{\omega \in Paths^{\mathcal{M}} \mid \omega \models \Psi\} \bowtie p \\ \omega \models X\Phi & \text{iff } \omega = s_1 t_1 s_2 \ldots \wedge\, s_2 \models \Phi \\ \omega \models \Phi_1 U^I \Phi_2 & \text{iff } \exists t \in I.(\forall t' \in [0,t).\, \omega@t' \models \Phi_1) \wedge \omega@t \models \Phi_2 \end{array}$$

1. Steady-state properties only depend on the first moment of the distributions of the times spent in the CTMC states, so they are already computed accurately by existing CTMC analysis techniques.

where a formal definition for the probability measure Pr_s on paths starting in state s is available in [32], [38]. Note how according to these semantics [38], until path formulae $\Phi_1 U^I \Phi_2$ are satisfied by a path ω if and only if Φ_2 is satisfied at some time instant t in the interval I and Φ_1 holds at all previous time instants t', i.e., for all $t' \in [0,t)$. Finally, a state s satisfies a steady-state formula $\mathcal{S}_{\bowtie p}[\Phi]$ iff, having started in state s, the probability of the CTMC being in a state where Φ holds *in the long run* satisfies the bound '$\bowtie p$'.

The shorthand notation $\Phi_1 U \Phi_2 \equiv \Phi_1 U^{[0,\infty)} \Phi_2$ and $F^I \Phi \equiv \mathsf{true}\ U^I \Phi$ is used when $I = [0,\infty)$ in an until formula and when the first part of an until formula is true, respectively. Probabilistic model checkers also support CSL formulae in which the bound '$\bowtie p$' from $P_{\bowtie p}[\Psi]$ is replaced with '$=?$', to indicate that the computation of the actual bound is required. We distinguish between the probability $Pr_s\{\omega \in Paths^{\mathcal{M}} \mid \omega \models \Psi\}$ that Ψ is satisfied by the paths starting in a state s, and the probability

$$\begin{array}{l} P_{=?}[\Psi] = \sum_{s \in S} \boldsymbol{\pi}(s) Pr_s\{\omega \in Paths^{\mathcal{M}} \mid \omega \models \Psi\} \\ \qquad\quad = Pr_{\boldsymbol{\pi}}\{\omega \in Paths^{\mathcal{M}} \mid \omega \models \Psi\} \end{array}$$

that Ψ is satisfied by the CTMC. In the analysis of system-level QoS properties, we are interested in computing the latter probability.

2.3 Phase-type distributions

OMNI uses *phase-type distributions* (PHDs) to refine the relevant elements of the analysed high-level abstract CTMC. PHDs model stochastic processes where the event of interest is the time to reach a specific state, and are widely used in the performance modelling of systems from domains ranging from call centres to healthcare [43], [44], [45]. PHDs support efficient numerical and analytical evaluation [21], and can approximate arbitrarily close any continuous distribution with a strictly positive density in $(0,\infty)$ [46], although PHD fitting of distributions with deterministic delays requires extremely large numbers of states.

A PHD is defined as the distribution of the time to absorption in a CTMC with one absorbing state [21]. The $N \geq 1$ transient states of the CTMC are called the *phases* of the PHD. With the possible reordering of states, the transition rate matrix of this CTMC can be expressed as:

$$\mathbf{R} = \left[\begin{array}{c|c} \mathbf{D}_0 & \mathbf{d_1} \\ \hline \mathbf{0} & 0 \end{array}\right], \quad (4)$$

where the $N \times N$ sub-matrix $\mathbf{D}_0$ specifies only transition rates between transient states, $\mathbf{0}$ is a $1 \times N$ row vector of zeros, and $\mathbf{d}_1$ is an $N \times 1$ vector whose elements specify the transition rates from the transient states to the absorbing state. The elements from each row of $\mathbf{R}$ add up to zero (cf. Definition 1), so we additionally have $\mathbf{D_0 1} + \mathbf{d_1} = \mathbf{0}$, where $\mathbf{1}$ and $\mathbf{0}$ are column vectors of N ones and N zeros, respectively. Thus, $\mathbf{d_1} = -\mathbf{D_0 1}$ and the PHD associated with this CTMC is fully defined by the sub-matrix $\mathbf{D}_0$ and the row vector $\boldsymbol{\pi}_0$ containing the first N elements of the initial probability vector $\boldsymbol{\pi}$ (as in most practical applications, we are only interested in PHDs that are acyclic and that cannot start in the absorbing state). We use the notation $\mathsf{PHD}(\boldsymbol{\pi}_0, \mathbf{D}_0)$ for this PHD.

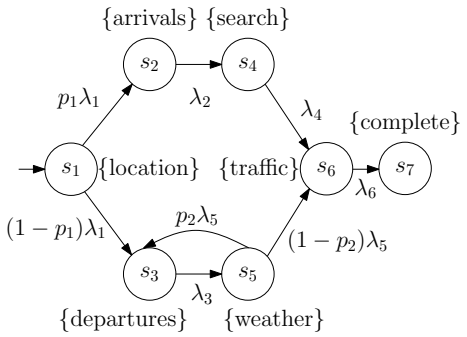


Fig. 2: High-level abstract CTMC modelling the handling of a request by the web application

2.4 Erlang distributions

The Erlang distribution [47] is a form of PHD in which k exponential phases, each with the same rate parameter λ, are placed in series. The Erlang distribution has a $(k+1)$-element initial probability vector $\boldsymbol{\pi} = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}$, such that the system always starts in an initial state s_0 and successively traverses states $s_1, s_2, \dots$ until it reaches an absorbing state s_k. The distribution represents the expected time to reach the absorbing state, and has the cumulative distribution function

$$F(k, \lambda, x) = 1 - \sum_{i=0}^{k-1} \frac{(\lambda x)^i}{i!} e^{-\lambda x}, \tag{5}$$

for $x \geq 0$, the mean $m = \frac{k}{\lambda}$, and the variance $\frac{k}{\lambda^2} = \frac{m^2}{k}$ (which approaches zero as $k \to \infty$).

3 Motivating example: QoS analysis of a web application

To illustrate the limitations of traditional CTMC-based QoS analysis, we consider a travel web application that handles two types of requests:

1. Requests from users who plan to meet and entertain a visitor arriving by train.
2. Requests from users looking for a possible destination for a day trip by train.

The handling of these requests by the application is modelled by the high-level abstract CTMC from Fig. 2, which can be obtained from a UML activity diagram of the application. The method for obtaining a Markov chain from an activity diagram is described in detail in [16], [20], [48], [49]. This method requires annotating the outgoing edges of decision nodes from the diagram with the probabilities with which these edges are taken during the execution of the modelled application. Markov model states are then created for each of the activities, decision and start/end nodes in the diagram, and state transitions are added for each edge between these nodes; the transitions corresponding to outgoing edges of decision nodes "inherit" the probabilities that annotate these edges, while all other transition probabilities have a value of 1.0.

The initial state s_1 of the CTMC from Fig. 2 corresponds to finding the location of the train station. For the first request type, which is expected to occur with probability p_1, this is followed by finding the train arrival time (state s_2), identifying suitable restaurants in the area (state s_4), obtaining a traffic report for the route from the user's location to the station (state s_6), and returning the response to the user (state s_7).

For the second request type, which occurs with probability $1 - p_1$, state s_1 is followed by finding a possible destination (state s_3), and obtaining a weather forecast for this destination (state s_5). With a probability of p_2 the weather is unsuitable and a new destination is selected (back to state s_3). Once a suitable destination is selected, the traffic report is obtained for travel to the station (state s_6) and the response is returned to the user (state s_7).

The component execution rates λ_1 to λ_6 depend on the implementations used for these components, and we consider that a team of software engineers wants to decide if the real web services from Table 1 are suitable for building the application. If they are suitable, the engineers need:

1. To select appropriate request-handling times to be specified in the application service-level agreement (SLA);
2. To choose a pricing scheme for the application.

Accordingly, the engineers want to assess several QoS properties of the travel application variant built using these publicly available web services:

P1 The probability of successfully handling user requests in under T seconds, for $0 < T \leq 4$.
P2 The probability of successfully handling "day trip" requests in under T seconds, for $0 < T \leq 4$.
P3 The expected profit per request handled, assuming that 1 cent is charged for requests handled within T seconds and a 2-cent penalty is paid for requests not handled within 3 seconds, for $0 < T \leq 3$.

Service response times are assumed exponentially distributed in QoS analysis based on CTMC (as well as queueing network) models. Therefore, the engineers use observed service execution times $t_{i1}, \dots, t_{in}$ for service i to estimate the service rate λ_i as

$$\lambda_i = \left(\frac{t_{i1} + t_{i2} + \dots + t_{in}}{n} \right)^{-1}. \tag{6}$$

These execution times can be taken from existing logs (e.g. of other applications that use the same services) or can be obtained through testing the web services individually. Finally, a probabilistic model checker is used to analyse properties **P1**–**P3** of the resulting CTMC. For this purpose, the three properties are first formalised as transient-state CSL formulae:

$$\begin{array}{ll} \textbf{P1} & P_{=?}[F^{[0,T]}\, complete] \\ \textbf{P2} & P_{=?}[\neg arrivals\ U^{[0,T]}\, complete]/(1-p_1) \\ \textbf{P3} & P_{=?}[F^{[0,T]}\, complete] - 2 \cdot P_{=?}[F^{(3,\infty)}\, complete] \end{array} \tag{7}$$

The value of T to be specified in the SLA is unknown a priori and hence we evaluate each property for a range of T values where $0 < T \leq 4$ for **P1** and **P2**, and $0 < T \leq 3$ for **P3**.

To replicate this process, we implemented a prototype version of the application and we used it to handle 270 randomly generated requests for $p_1 = 0.3$ and $p_2 = 0.1$.

TABLE 1: Web services considered for the web application

Label	Thid-party service	URL	rate (s^{-1})
location	Bing location service	http://dev.virtualearth.net/REST/v1/Locations	9.62
arrivals	Thales rail arrival board	http://www.livedepartureboards.co.uk/ldbws/	19.88
departures	Thales rail departures board	http://www.livedepartureboards.co.uk/ldbws/	19.46
search	Bing web search	https://api.datamarket.azure.com/Bing/Search	1.85
weather	WebserviceX.net weather service	http://www.webservicex.net/globalweather.asmx	1.11
traffic	Bing traffic service	http://dev.virtualearth.net/REST/v1/Traffic	2.51

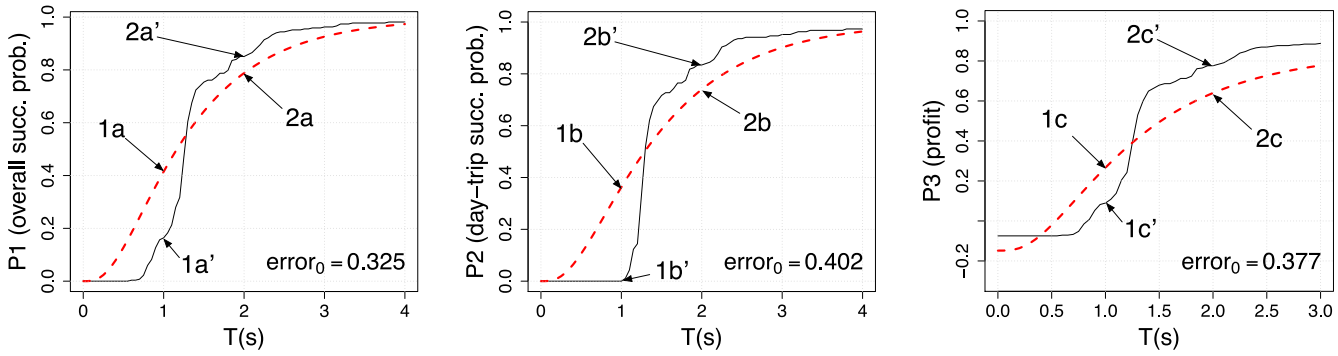


Fig. 3: Predicted (dashed lines) versus actual (continuous lines) property values

Obtaining transition probabilities for Markov chains from real-world systems, and the effects of transition probabilities on system performance, have previously been considered [50], [51]. To decouple these effects from those due to the temporal characteristics of component behaviours, we utilise fixed probabilities for our motivating example. However, for the second system used to evaluate OMNI (Section 6.1) we extract the transition probabilities from system logs, showing that OMNI also provides significant improvements in verification accuracy in this setting. We obtained sample execution times for each web service (between 81 for arrivals and search and 270 for location and traffic), and we applied (6) to these observations, calculating the estimate service rates from Table 1. Note that these observations are equivalent to observations obtained from unit testing the six services separately. This is due to the statistical independence of the execution times of different services, which we confirmed by calculating the Pearson correlation coefficient of the observations for every pair of services – the obtained coefficient values, between -0.17 and $+0.11$, indicate lack of correlation. We then used the model checker PRISM [14] to analyse the CTMC for these rates, and thus to predict the values of properties (7).

To assess the accuracy of the predictions, we also calculated the actual values of these properties at each time value T using detailed timing information logged by our application. The error associated with a single property evaluation may be quantified as the absolute difference between actual and predicted values

$$|actual(T) - predicted(T)| \tag{8}$$

The predictions obtained through CTMC analysis and the actual property values across the range of T values are compared in Fig. 3. The errors reported in the figure are calculated using the distance measure recommended for assessing the overall error of CTMC/PHD model fitting in [21], [52], [53], [54], i.e., the area difference between the actual and the predicted property values:

$$error = \int_0^{T_{\max}} |actual(T) - predicted(T)|\ dT, \tag{9}$$

where $T_{\max} = 4$ for properties **P1** and **P2**, and $T_{\max} = 3$ for property **P3**.[2] Later in the paper, we will use this error measure to assess the improvements in accuracy due to the OMNI model refinement. In this section we focus on the limitations of CTMC-based transient analysis. Therefore, recall that the software engineers must make their decisions based only on the predicted property values from Fig. 3; two of these decisions and their associated scenarios are described below.

Scenario 1. The engineers note that:

(a) the predicted overall success probability (property **P1**) at T=1s is 0.415 (marked 1a in Fig. 3), i.e., slightly over 40% of the requests are predicted to be handled within 1s;

(b) the predicted day-trip success probability (property **P2**) at $T = 1$s is 0.363 (1b in Fig. 3), i.e., over 36% of the day-trip requests are predicted to be handled within 1s;

(c) the expected profit (property **P3**) at $T = 1$s, i.e., when charging 1 cent for requests handled within 1s, is 0.27 cents (1c in Fig. 3).

Accordingly, the engineers decide to use the services from Table 1 to implement the travel web application, with an SLA "promising" that requests will be handled within 1s

2. Both underestimation and overestimation of QoS property values contribute to the error because both can lead to undesirable false positives or false negatives when assessing whether QoS requirements are met. For example, overestimates of the overall success probability of a system can falsely indicate that a requirement that places a lower bound on this probability is met and the system is safe to use (false negative), while underestimates of the same property can falsely indicate that the requirement is violated and the system should not be used (false positive).

with 0.4 success probability, "day trip" requests will be handled within 1s with 0.35 success probability, and charging 1 cent for requests handled within 1s. As shown in Fig. 3, the actual property values at $T = 1$s are 0.164 for **P1** (marked 1a′ in Fig. 3), 0 for **P2** (1b′ in Fig. 3) and 0.09 cents for **P3** (1c′ in Fig. 3), so this decision would be wrong – both promises would be violated by a wide margin, and the actual profit would be under a third of the predicted profit.

Scenario 2. The engineers observe that the success probabilities of handling requests or "day trip" requests within 2s are below 0.8 – the predicted values for properties **P1** and **P2** at $T = 2$s are 0.79 (2a in Fig. 3) and 0.74 (2b in Fig. 3), respectively; and/or that the expected profit is below 0.7 cents per request when charging 1 cent for each request handled within 2s (2c in Fig. 3). As such, they decide to look for alternative services for the application. As shown by points 2a′–2c′ in Fig. 3, all the constraints underpinning this decision are actually satisfied, so the decision would also be wrong.

We chose the times and constrains in the two hypothetical decisions to show how the current use of idealised CTMC models in QoS analysis *may* yield invalid decisions. The fact that choosing different times and constrains could produce valid decisions is not enough: engineering decisions are meant to be consistently valid, not down to chance. It is this major limitation of traditional CTMC-based QoS analysis that our CTMC refinement method addresses as described in the next section.

4 The OMNI method for CTMC refinement

4.1 Overview

OMNI addresses the refinement of high-level CTMC models $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ of software systems that satisfy the following assumptions:

- Each state $s_i \in S$ corresponds to a component of the system, and $\boldsymbol{\pi}(s_i)$ is the probability that s_i is the initial component executed by the system;
- For any distinct states from $s_i, s_j \in S$, the transition rate $\mathbf{R}(s_i, s_j) = p_{ij}\lambda_i$, where p_{ij} represents the (known or estimated) probability (2) that component i is followed by component j and λ_i is obtained by applying (6) to n_i observed execution times $\tau_{i1}, \tau_{i2}, \ldots, \tau_{in_i}$ of component i;
- Each state $s_i \in S$ is labelled with the name of its corresponding component, which we will call "component i" for simplicity.

This CTMC model makes the standard assumption that component execution times are exponentially distributed. However, this assumption is typically invalid for two reasons. First, each component i has a *delay* δ_i (i.e. minimum execution time) approximated by

$$\delta_i \approx \min_{j=1}^{n_i} \tau_{ij} \tag{10}$$

such that its probability of completion within $\delta(s_i)$ time units is zero. In contrast, modelling the execution time of the component as exponentially distributed with rate λ_i yields a non-zero probability $1 - e^{-\lambda_i \delta_i}$ of completion within δ_i time units. Second, even the *holding times*

$$\tau'_{i1} = \tau_{i1} - \delta_i,\ \tau'_{i2} = \tau_{i2} - \delta_i,\ \ldots,\ \tau'_{in_i} = \tau_{in_i} - \delta_i \tag{11}$$

of the component are rarely exponentially distributed.

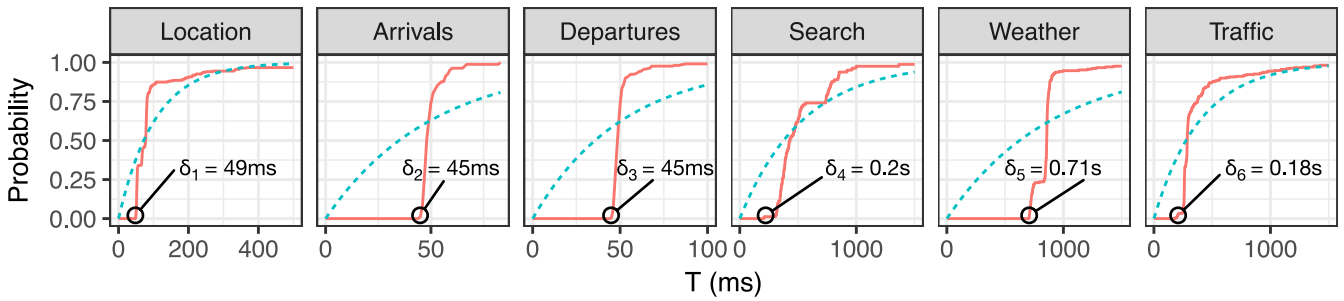
Example 1. Fig. 4a shows the empirical cumulative distribution functions (CDFs) for the execution times of the six services from our motivating example (cf. Table 1), and the associated exponential models with rates given by (6). The six services have minimum observed execution times δ_1 to δ_6 between 45ms and 0.71s (due to network latency and request processing time), and their exponential model is a poor representation of the observed temporal behaviour. Furthermore, the best-fit exponential model of the observed holding times for these services (shown in Fig. 4b) is also inaccurate.

OMNI overcomes these significant problems by generating a refined CTMC for each QoS property of interest in two steps, and uses standard probabilistic model checking to analyse the refined CTMC. As shown in Fig. 5, the first OMNI step, called *component classification*, partitions the states of the high-level CTMC into subsets that require different types of refinement because of the different impact of their associated system components on the analysed property. For instance, components unused on an execution path have no effect on QoS properties (e.g. response time) associated solely with that path, and therefore their corresponding states from the high-level CTMC need not be refined. The second OMNI step, called *selective refinement*, replaces the states which correspond to components that impact the analysed property with new states and transitions that model the delays and holding times of these components by means of Erlang distributions [47] and phase-type distributions (PHDs) [21], respectively.

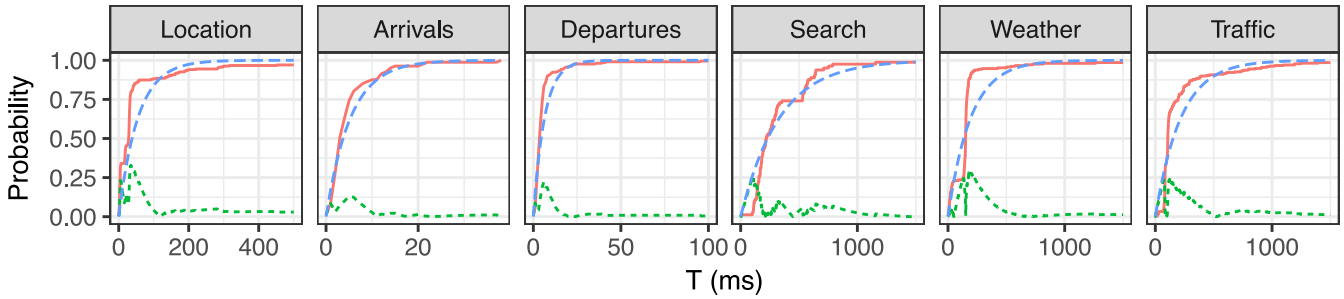
As shown by our experimental results from Section 6, the two-step OMNI process produces refined CTMCs that are often much smaller and faster to analyse than the CTMCs obtained by obliviously refining every state of the high-level CTMC, e.g. as done in our preliminary work from [31]. These benefits dominate the slight disadvantage of having to refine the high-level CTMC for each analysed property, which is further mitigated by our OMNI tool by caching and reusing refinement results across successive refinements of the same high-level CTMC, as described in Section 5. Likewise, modelling the delay and holding time of system components separately (rather than using single-PHD fitting) yields smaller and more accurate refined models, in line with existing theory [46] and our preliminary results from [31].

Several factors can impede or impact the success of our OMNI method:

1. Components with execution times that are not statistically independent. Markov models assume that the transition rates associated with different states are statistically independent. If the execution times of different components are not independent (e.g., because the components are running on the same server), then this premise is not satisfied, and OMNI cannot be applied.



(a) Empirical CDF for the service execution times (continuous lines) versus exponential models with rates computed from observed data (dashed lines)



(b) Empirical CDF for the service holding times (continuous lines) versus exponential models with rates computed from observed holding times (long dashed lines); for all services except Arrivals the difference between the two (short dashed lines) exceeds 20% for multiple values of T

Fig. 4: The services from the motivating example have non-zero delays and non-exponentially distributed holding times

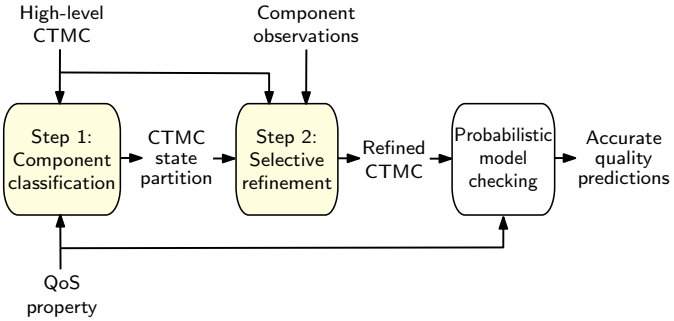


Fig. 5: OMNI CTMC refinement and verification

2. Changing component behaviour. If the system components change their behaviour significantly over time, then OMNI cannot predict the changed behaviour. This is a more general difficulty with model-based prediction.
3. Insufficient observations of component execution times. The accuracy of OMNI-refined models decreases when fewer observations of the system components are available. We provide details about the impact of the training dataset size on the OMNI accuracy in Section 6.4.

The component classification and selective refinement steps of OMNI are presented in the rest of this section.

4.2 Component classification

Given a high-level CTMC model $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ of a system, and a QoS property encoded by the transient CSL formula $P_{=?}[\Phi_1 U^I \Phi_2]$, this OMNI step builds a partition

$$S = S_? \cup S_0 \cup S_1 \cup S_2 \cup \ldots \cup S_m \tag{12}$$

of the state set S. Intuitively, the "eXclude-from-refinement" set $S_?$ will contain states with zero probability of occurring on paths that satisfy $\Phi_1 U^I \Phi_2$; the "Once-only" set S_0 will contain states with probability 1.0 of appearing once and only once on every path that satisfies $\Phi_1 U^I \Phi_2$; and each "together" set S_i will contain states that can only appear as a sequence on paths that satisfy $\Phi_1 U^I \Phi_2$. Formal definitions of the disjoint sets $S_?$, S_0, and S_1 to S_m and descriptions of their roles in OMNI are provided in Sections 4.2.1–4.2.3.

4.2.1 *Exclude-from-refinement state sets*

Definition 4.1. The *exclude-from-refinement state set* $S_?$ associated with an until path formula $P_{=?}[\Phi_1 U^I \Phi_2]$ over the continuous-time Markov chain $\mathcal{M} = \mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ is the set of CTMC states

$$S_? = \{s \in S \mid P_{=?}[(\neg s \wedge \Phi_1) U \Phi_2] = P_{=?}[\Phi_1 U \Phi_2]\}, \tag{13}$$

where, for each state $s \in S$, AP is extended with an atomic proposition also named 's' that is true in state s and false in every other state. Thus, $S_?$ comprises all states s for which the probability $P_{=?}[(\neg s \wedge \Phi_1) U \Phi_2]$ of reaching a state satisfying Φ_2 along paths that *do not contain state* s and on which Φ_1 holds in all preceding states is the same as the probability $P_{=?}[\Phi_1 U \Phi_2]$ of reaching a state that satisfies Φ_2 along paths on which Φ_1 holds in all preceding states.

Theorem 1. Let $S_?$ be the exclude-from-refinement state set associated with the until path formula $P_{=?}[\Phi_1 U^I \Phi_2]$ over

the continuous-time Markov chain $\mathcal{M} = \mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ with atomic proposition set AP. Then, for any $I \subseteq \mathbb{R}_{\geq 0}$, the probability $P_{=?}[\Phi_1 U^I \Phi_2]$ does not depend on the transition times from states in $S_?$.

Proof. The proof is by contradiction. Consider a generic state $s_? \in S_?$ and the following sets of paths:

$$A = \{\omega \in Paths^{\mathcal{M}} \mid \exists t > 0 \,.\, (\omega@t \models \Phi_2 \wedge (\forall t' \in [0,t).\omega@t' \models \Phi_1))\}$$
$$B = \{\omega \in Paths^{\mathcal{M}} \mid \exists t > 0 \,.\, (\omega@t \models \Phi_2 \wedge (\forall t' \in [0,t).\omega@t' \models \Phi_1 \wedge \omega@t' \neq s_?))\}$$
$$C = \{\omega \in Paths^{\mathcal{M}} \mid \exists t > 0 \,.\, (\omega@t \models \Phi_2 \wedge (\forall t' \in [0,t).\omega@t' \models \Phi_1) \wedge (\exists t' \in [0,t).\omega@t' = s_?))\}$$

As $A = B \cup C$ and $B \cap C = \emptyset$, we have $Pr_{\boldsymbol{\pi}}(A) = Pr_{\boldsymbol{\pi}}(B) + Pr_{\boldsymbol{\pi}}(C)$. However, according to (13), $Pr_{\boldsymbol{\pi}}(A) = P_{=?}[\Phi_1 U \Phi_2] = P_{=?}[(\neg s \wedge \Phi_1) U \Phi_2] = Pr_{\boldsymbol{\pi}}(B)$, so $Pr_{\boldsymbol{\pi}}(C) = 0$.

Assume now that the time spent by the CTMC in state $s_?$ has an impact on the value of $P_{=?}[\Phi_1 U^I \Phi_2]$ over $Paths^{\mathcal{M}}$ for an interval $I \subseteq \mathbb{R}_{\geq 0}$. This requires that, at least for some (possibly very small) values of the time $t_? > 0$ spent in $s_?$, $s_?$ appears on paths from a set

$$C' = \{\omega \in Paths^{\mathcal{M}} \mid \exists t \in I \,.\, (\omega@t \models \Phi_2 \wedge (\forall t' \in [0,t).\omega@t' \models \Phi_1) \wedge (\exists t' \in [0,t).\omega@t' = s_?))\}$$

such that $Pr_{\boldsymbol{\pi}}(C') > 0$; otherwise, varying $t_?$ cannot have any impact on

$$P_{=?}[\Phi_1 U^I \Phi_2] = Pr_{\boldsymbol{\pi}}\{\omega \in Paths^{\mathcal{M}} \mid \exists t \in I \,.\, (\omega@t \models \Phi_2 \wedge (\forall t' \in [0,t).\omega@t' \models \Phi_1))\}$$

However, since $C' \subseteq C$ we must have $Pr_{\boldsymbol{\pi}}(C) \geq Pr_{\boldsymbol{\pi}}(C') > 0$, which contradicts our earlier finding that $Pr_{\boldsymbol{\pi}}(C) = 0$, completing the proof. □

Theorem 1 allows OMNI to leave the states from $S_?$ unrefined with no loss of accuracy in the QoS analysis results. The theorem also provides a method for obtaining $S_?$ by computing the until formula $P_{=?}[(\neg s \wedge \Phi_1) U \Phi_2]$ for each state s of the high-level CTMC (i.e. for each system component) and comparing the result with the value of the CSL formula $P_{=?}[\Phi_1 U \Phi_2]$, which is only computed once. Existing probabilistic model checkers compute these *unbounded* until formulae very efficiently, as they only depend on the probabilities (2) of transition between CTMC states and not on the state transition rates [32], [38].[3]

Example 2. Consider the QoS properties (7) of the web application from our motivating example. For property **P2** and the high-level CTMC model from Fig. 2, we have

$$P_{=?}[\neg arrivals \; U \; complete] = 1 - p_1 = P_{=?}[(\neg s_2 \wedge \neg arrivals) \; U \; complete] = P_{=?}[(\neg s_4 \wedge \neg arrivals) \; U \; complete] = P_{=?}[(\neg s_7 \wedge \neg arrivals) \; U \; complete],$$

(and $P_{=?}[(\neg s \wedge \neg arrivals) \, U \, complete] \neq 1 - p_1$ for any other state s), so $S_? = \{s_2, s_4, s_7\}$ for **P2**. Applying Theorem 1 to the other two properties from (7) yields $S_? = \{s_7\}$.

3. To asses the time taken by model checking, an experiment was carried out to evaluate each state from the motivating example for inclusion in $S_?$. This experiment was repeated 30 times and the average time taken by model checking each state was found to be 1.6ms.

4.2.2 Once-only state sets

Definition 4.2. The *once-only state set* S_{O} associated with an until path formula $P_{=?}[\Phi_1 U^I \Phi_2]$ over the continuous-time Markov chain $\mathcal{M} = \mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ is the set

$$S_{\mathsf{O}} = \{s \in S \setminus S_? \mid P_{>0}[\Phi_1 U \Phi_2] \wedge P_{\leq 0}[(\neg s \wedge \Phi_1) U \Phi_2] \wedge \forall s' \in S \,.\, (s \models P_{>0}[X s'] \rightarrow s' \models P_{\leq 0}[\neg S_? U s])\}, \tag{14}$$

where the until formula $\neg S_? U s$ holds for paths that reach state s without going through any states from $S_?$ (which corresponds to labelling the states from $S_?$ with the atomic proposition '$S_?$').

The next theorem asserts that for every state s_{O} from S_{O}, $P_{=?}[\Phi_1 U^I \Phi_2]$ can be calculated by applying the probability measure $Pr_{\boldsymbol{\pi}}$ to the set of paths ω which, in addition to satisfying the clause specified by the CSL semantics (i.e., $\exists t \in I.(\forall t' \in [0,t). \, \omega@t' \models \Phi_1) \wedge \omega@t \models \Phi_2$), contain s_{O} once and only once *before time instant* t. Using the unique existential quantifier $\exists!$, the last clause can be formalised as $\exists! i \,.\, (\omega[i] = s_{\mathsf{O}} \wedge \sum_{j=1}^{i} t_j < t)$, where t_j is the time spent in the j-th state on the path (cf. Section 2.1).

Theorem 2. Let S_{O} be the once-only state set associated with the until path formula $P_{=?}[\Phi_1 U^I \Phi_2]$ over the continuous-time Markov chain $\mathcal{M} = \mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$. Then, for any state $s_{\mathsf{O}} \in S_{\mathsf{O}}$ and interval $I \subseteq \mathbb{R}_{\geq 0}$,

$$P_{=?}[\Phi_1 U^I \Phi_2] = Pr_{\boldsymbol{\pi}}\{\omega \in Paths^{\mathcal{M}} \mid \exists t \in I \,.\, (\omega@t \models \Phi_2 \wedge (\forall t' \in [0,t) \,.\, \omega@t' \models \Phi_1) \wedge \exists! i \,.\, (\omega[i] = s_{\mathsf{O}} \wedge \textstyle\sum_{j=1}^{i} t_j < t))\}. \tag{15}$$

Proof. Let A' denote the subset of $Paths^{\mathcal{M}}$ from (15). According to CSL semantics, $P_{=?}[\Phi_1 U^I \Phi_2] = Pr_{\boldsymbol{\pi}}(A)$ where

$$A = \{\omega \in Paths^{\mathcal{M}} \mid \exists t \in I \,.\, (\omega@t \models \Phi_2 \wedge (\forall t' \in [0,t).\omega@t' \models \Phi_1))\}.$$

Since $A' \subseteq A$, we have $P_{=?}[\Phi_1 U^I \Phi_2] = Pr_{\boldsymbol{\pi}}(A) = Pr_{\boldsymbol{\pi}}(A') + Pr_{\boldsymbol{\pi}}(A \setminus A')$, so to prove the theorem we must show that $Pr_{\boldsymbol{\pi}}(A \setminus A') = 0$. To this end, we partition $A \setminus A'$ into two disjoint subsets: A_1, comprising the paths that do not contain state s_{O} before time t from the first line of (15), and A_2, comprising the paths that contain state s_{O} before time t more than once. Since $P_{\leq 0}[(\neg s_{\mathsf{O}} \wedge \Phi_1) U \Phi_2]$ holds (according to the definition of S_{O}), $Pr_{\boldsymbol{\pi}}(A_1) = 0$. Similarly, since $\forall s' \in S \,.\, (s_{\mathsf{O}} \models P_{>0}[X s'] \rightarrow s' \models P_{\leq 0}[\neg S_? U s_{\mathsf{O}}])$ holds, the set of paths satisfying $\Phi_1 U^I \Phi_2$ and containing s_{O} twice (without reaching states in $S_?$) occur with probability zero. As A_2 is included in this set, we necessarily have $Pr_{\boldsymbol{\pi}}(A_2) = 0$. We conclude that $Pr_{\boldsymbol{\pi}}(A \setminus A') = Pr_{\boldsymbol{\pi}}(A_1) + Pr_{\boldsymbol{\pi}}(A_2) = 0$, which completes the proof. □

OMNI exploits Theorem 2 in two ways. First, since S_{O} states correspond to system components always executed before $\Phi_1 U^I \Phi_2$ becomes true, $P_{=?}[\Phi_1 U^I \Phi_2] = 0$ for any interval $I \subseteq \left[0, \sum_{s_i \in S_{\mathsf{O}}} \delta_i\right)$, where δ_i is the delay (10) of the component i associated with state s_i. Therefore, OMNI returns a zero probability in this scenario without performing probabilistic model checking. Second, because the components associated with S_{O} states are executed precisely once on relevant CTMC paths, no modelling of their delays is required, and OMNI only needs to model the

holding times of these states. Importantly, obtaining $S_?$ to enable these simplifications only requires the probabilities of unbounded until and next path formulae (cf. (14)), which probabilistic model checkers can compute efficiently for the reasons we explained earlier in this section.

Example 3. Consider property **P1** from the QoS properties (7) in our motivating example: $P_{=?}[F^{[0,T]}\,complete]$. In line with definition (14), we obtain the set $S_?$ for this property by first evaluating the following CSL formulae for the high-level CTMC from Fig. 2:

- $P_{>0}[true\,U\,complete]$ which holds as $P_{=?}[F\,complete]=1$
- $P_{\leq 0}[(\neg s \wedge true)\,U\,complete]=P_{\leq 0}[\neg s\,U\,complete]$, which holds only for states s_1 and s_6.

The constraint $\forall s' \in S.(s \models P_{>0}[X\,s'] \rightarrow s' \models P_{\leq 0}[\neg S_? U\, s])$ is then checked only for the $S_?$-candidate states $s = s_1$ and $s = s_6$, taking into account the fact that $S_? = \emptyset$ (cf. Example 2). For instance, since $s_1 \models P_{>0}[X\,s']$ only for $s' \in \{s_2, s_3\}$, and $s_2, s_3 \models P_{\leq 0}[true\ U\ s_1]$, we conclude that $s_1 \in S_?$. Similarly, $s_6 \models P_{>0}[X\,s']$ only if $s' = s_7$ and $s_7 \models P_{\leq 0}[true\ U\ s_6]$, so $s_6 \in S_?$, giving $S_? = \{s_1, s_6\}$. It is easy to show that the same "once-only" state set is obtained for the other two properties from (7).

4.2.3 Together state sets

Finally, the result in this section supports the calculation and exploitation of the "together" state sets from (12).

Definition 4.3. The *together state sets* $S_1, S_2, \ldots, S_m$ for an until path formula $P_{=?}[\Phi_1 U^I \Phi_2]$ over the Markov chain $\mathcal{M} = \mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ are the state sets comprising the same elements as the m state sequences returned by function TOGETHERSEQS($\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$, $S_?$, $S_?$) from Algorithm 1, where $S_?$ and $S_?$ are the exclude-from-refinement and once-only state sets for the formula.

The function TOGETHERSEQS builds the m state sequences in successive iterations of its outer while loop (lines 3–26). The set *States* maintains the states yet to be allocated to sequences (initially $S \setminus (S_? \cup S_?)$, cf. line 2), and each new sequence T starts with a single element picked randomly from *States* (line 4). The inner while loop in lines 7–24 "grows" this sequence. First, the if statement in lines 8–15 tries to grow the sequence to the left with a state s that "precedes" the sequence, in the sense that the only outgoing CTMC transition from s is to the sequence head, and the only way of reaching the sequence head is through an incoming CTMC transition from s. Analogously, the if statement in lines 16-23 grows the sequence to the right, by appending to it the state that "succeeds" the state at the tail of the sequence, if such a "successor" state exists. The predecessor and successor states of a state s are computed by the functions PRED and SUCC, respectively, where these functions return NIL if the states they attempt to find do not exist. The inner while loop terminates when the *States* set becomes empty or the sequence T has no more predecessors or successors, so the flags *left* and *right* are set to *false* in lines 13 and 21, respectively. On exit from this while loop, the sequence T is added to the set of sequences TS, which is returned (line 27) after the outer while loop also terminates when *States* becomes empty. Termination is guaranteed since at least one element is removed from *States* in each iteration of this while loop (in line 5).

To analyse the complexity of TOGETHERSEQS, we note that the worst case scenario corresponds to $S_? = S_? = \emptyset$ and to the function returning only sequences of length 1, in which case the outer while loop is executed $|S|$ times with both PRED and SUCC invoked once in each iteration.

Algorithm 1 Generation of "together" state sequences

```
1: function TogetherSeqs(CTMC(S, π, R), S_?, S_?)
2:     TS ← ∅, States ← S \ (S_? ∪ S_?)
3:     while States ≠ ∅ do
4:         s ← PickAnyElement(States)
5:         T ← ⟨s⟩, States ← States \ {s}
6:         left, right ← true
7:         while (left ∨ right) ∧ States ≠ ∅ do
8:             if left then
9:                 s ← Pred(Head(T), States, S, π, R)
10:                if s ≠ NIL then
11:                    T ← ⟨s⟩⌢T, States ← States \ {s}
12:                else
13:                    left ← false
14:                end if
15:            end if
16:            if right then
17:                s ← Succ(Tail(T), States, S, π, R)
18:                if s ≠ NIL then
19:                    T ← T⌢⟨s⟩, States ← States \ {s}
20:                else
21:                    right ← false
22:                end if
23:            end if
24:        end while
25:        TS ← TS ∪ {T}
26:    end while
27:    return TS
28: end function

29: function Pred(s, States, S, π, R)
30:    if π(s) > 0 then return NIL end if
31:    for s' ∈ States do
32:        if R(s', s) > 0 ∧ ∀s'' ∈ S \ {s, s'} .
                  (R(s', s'') = 0 ∧ R(s'', s) = 0) then
33:            return s'
34:        end if
35:    end for
36:    return NIL
37: end function

38: function Succ(s, States, S, π, R)
39:    for s' ∈ States do
40:        if π(s') = 0 ∧ R(s, s') > 0 ∧ ∀s'' ∈ S \ {s, s'} .
                  (R(s, s'') = 0 ∧ R(s'', s') = 0) then
41:            return s'
42:        end if
43:    end for
44:    return NIL
45: end function
```

The if statements from PRED and SUCC perform $O(|S|)$ comparisons, and are executed within for loops with $O(|S|)$ iterations, yielding an $O(|S|^2)$ complexity for each function, and an overall $O(|S|^3)$ complexity for the algorithm.

Theorem 3. If $T = \langle s_{i1}, s_{i2}, \ldots, s_{iN_i} \rangle$ is one of the sequences returned by TOGETHERSEQS, ω a path that satisfies $\Phi_1 U^I \Phi_2$ for an interval $I \subseteq \mathbb{R}_{\geq 0}$, and $t \in I$ the earliest time when $\omega@t \models \Phi_2$ (with $\omega@t' \models \Phi_1$ for all $t' \in [0, t)$), then up to time t the states from T can only appear on ω as complete sequences $\ldots s_{i1} t_{i1} s_{i2} t_{i2} \ldots s_{iN_i} t_{iN_i} \ldots$.

Proof. The case $N_i = 1$ is trivial, so we assume $N_i > 1$ in the rest of the proof. We have two cases: either ω contains no states from T, or it contains at least one state from T. In the former case, the theorem is proven. In the latter case, consider any state s_{i_j} that occurs on ω, $1 \leq j \leq N_i$. The states s_{i1}, s_{i2}, …, $s_{i,j-1}$ must also occur on ω, in this order and just before s_{ij}, as transitioning through each of these states is the only way to reach s_{ij} in the CTMC. Moreover, $s_{i,j+1}$, $s_{i,j+2}$, …, s_{iN_i} must immediately follow s_{ij} on ω (in this order) because s_{ij} is not an absorbing state and its only outgoing transition is to $s_{i,j+1}$, etc. Hence, the path is of the form $\omega = s_1 t_1 s_2 t_2 \ldots s_x t_x s_{i1} t_{i1} s_{i2} t_{i2} \ldots s_{iN_i} t_{iN_i} \ldots$ for some $x \geq 0$. To prove that this occurrence of all states from T on ω is either up to or after time t, we show that it is not possible to have $\omega@t = s_{ij}$ for any $j < N$. Indeed, if we assume $\omega@t = s_{ij}$ then according to the hypothesis $s_{ij} \models \Phi_2$ must hold. As this must be true not only for ω but also for any other path ω' that satisfies $\Phi_1 U \Phi_2$ and contains the states from T, definition (13) implies that $s_{i,j+1}, s_{i,j+2}, \ldots, s_{iN_i} \in S_?$ because ω' comprises states that satisfy Φ_1 followed by state s_{ij} that satisfies Φ_2, followed by the $s_{i,j+1}, s_{i,j+2}, \ldots, s_{iN_i}$. However, having states from T in $S_?$ is not possible since line 2 of TOGETHERSEQS removes $S_?$ from the set of states used to generate T. □

Theorem 3 allows OMNI to model the delays of all states in the same "together" set S_i, $1 \leq i \leq m$, as a *joint delay*

$$\Delta_i = \sum_{j=1}^{N_i} \delta_{ij}, \tag{16}$$

since the relevant part of any path that influences the value of $P_{=?}[\Phi_1 U^I \Phi_2] = 0$ contains either all these states or none of them.

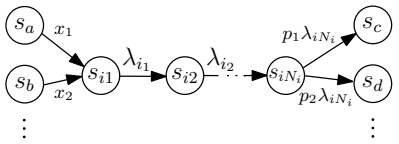
Example 4. Consider again the QoS properties (7) of the web application from our motivating example. For property **P2**, TOGETHERSEQS is called with $S_? = \{s_2, s_4, s_7\}$ (cf. Example 2) and $S_0 = \{s_1, s_6\}$ (cf. Example 3), so it starts with $States = \{s_1, s_2, \ldots, s_7\} \setminus (S_? \cup S_0) = \{s_3, s_5\}$ in line 2. Irrespective of which of s_3 and s_5 is picked in line 5, the other state will be added to the same "together" sequence since the two states always follow one another with no intermediate states. The "together" sets for the other two properties from (7) are given in Table 2, which brings together the results from Examples 2–4.

4.3 Selective refinement

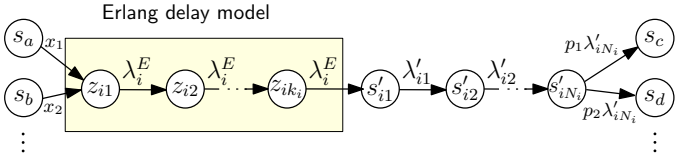
The second OMNI step models the delays and holding times of the relevant components of the analysed system according to the rules established in the previous section. These rules are summarised in Table 3, and require methods for joint delay modelling (for components associated with "together" CTMC states) and for individual holding time modelling (for components not associated with $S_?$ states). The two methods are described next.

TABLE 2: CTMC state partition for the web application properties

Property	$S_?$	S_0	$S_1, S_2, \ldots, S_m$
P1	$\{s_7\}$	$\{s_1, s_6\}$	$\{s_2, s_4\}, \{s_3, s_5\}$
P2	$\{s_2, s_4, s_7\}$	$\{s_1, s_6\}$	$\{s_3, s_5\}$
P3	$\{s_7\}$	$\{s_1, s_6\}$	$\{s_2, s_4\}, \{s_3, s_5\}$



(a) Model of "together" components $i1, i2, \ldots, iN_i$ in the high-level CTMC (showing generic incoming transitions of rates $x_1, x_2, \ldots$ from states $s_a, s_b, \ldots$ into state s_{i1}, and generic outgoing transitions of probabilities $p_1, p_2, \ldots$ and rate λ_{iN_i} from state s_{iN_i} to states $s_c, s_d, \ldots$)



(b) Model of components $i1, i2, \ldots, iN_i$ after joint delay modelling

Fig. 6: Joint delay modelling for "together" state set S_i whose final state s_{iN_i} has $M_i \geq 1$ outgoing transitions

4.3.1 Joint delay modelling

For each "together" set $S_i = \{s_{i1}, s_{i2}, \ldots, s_{iN_i}\}$, OMNI extends the CTMC with additional states and transitions that model the joint delay Δ_i from (16) by means of an Erlang distribution, i.e., a sum of several independent exponential distributions with the same rate [47]. As shown in Fig. 6, this involves replacing the states from S_i with a sequence of delay-modelling states z_{i1}, z_{i2}, …, z_{ik_i} that encode an Erlang-k_i distribution of rate λ_i^E, followed by states $s'_{i1}, s'_{i2}, \ldots, s'_{iN_i}$ with transitions matching those from the high-level CTMC but of rates λ'_{i1}, λ'_{i2}, …, λ'_{iN}.

However, delays are not modelled perfectly by Erlang distributions: for any *error* $\epsilon \in (0, 1)$, there is a (small) probability p that the refined CTMC leaves state z_{ik_i} within $\Delta_i(1-\epsilon)$ time units of entering z_{i1}. Given specific values for ϵ and p, the theorem below supports the calculation of the parameters k_i, λ_i^E and λ'_{i1} to λ'_{iN_i} for our joint delay modelling.

Theorem 4. Given an error bound $\epsilon \in (0, 1)$, if the joint delay modelling parameters k_i, λ_i^E and λ'_{i1} to λ'_{iN_i} satisfy

$$\begin{aligned} &\text{(a) } 1 - \textstyle\sum_{l=0}^{k_i-1} \frac{(k_i(1-\epsilon))^l e^{-k_i(1-\epsilon)}}{l!} = p \\ &\text{(b) } \lambda_i^E = \tfrac{k_i}{\Delta_i} \\ &\text{(c) } \forall j \in \{1, 2, \ldots, N_i\} \,.\, \lambda'_{ij} = \tfrac{\lambda_{ij}}{1 - \lambda_{ij}\delta_{ij}} \end{aligned} \tag{17}$$

TABLE 3: OMNI rules for modelling the delays and holding times of different types of system components

	S_{X} **components**	S_{O} **components**	S_i **components**, $1 \leq i \leq m$
delay	no modelling needed	no modelling needed†	joint delay modelling
holding time	no modelling needed	per component modelling	per component modelling

$^\dagger S_{\mathsf{O}}$ components introduce a deterministic delay $\sum_{s_i \in S_{\mathsf{O}}} \delta_i$

TABLE 4: Precomputed k_i values used in the experiments from Section 6

error ϵ	0.2	0.2		0.1	0.1
prob. p	0.29	0.08	$\cdots$	0.16	0.05
k_i	10	45		100	259

for some value $p \in (0,1)$ then the following properties hold for the refined CTMC:

(i) The probability that the CTMC leaves state z_{ik_i} within $\Delta_i(1-\epsilon)$ time units from entering state z_{i1} is p;

(ii) The expected time for the refined CTMC to leave s'_{iN_i} after entering state z_{i1} is $\sum_{j=1}^{N_i} \lambda_{ij}^{-1}$. This is also the expected time for the high-level CTMC to leave state s_{iN_i} after entering state s_{i1}, so the joint delay modelling preserves the first moment of the distribution associated with the refined CTMC states.

Proof. To prove (i), recall from Section 2.4 that the cumulative distribution function of an Erlang-k distribution with rate λ is $F(k,\lambda,x) = 1 - \sum_{l=0}^{k-1} \frac{(\lambda x)^l e^{-\lambda x}}{l!}$, so (17a) can be rewritten as $F(k_i, \lambda_i^E, \Delta_i(1-\epsilon)) = p$ since $k_i = \lambda_i^E \Delta_i$ according to (17b). Therefore, the probability that the Erlang delay model from Fig. 6 will transition from entering state z_{i1} to exiting state z_{ik_i} within $\Delta_i(1-\epsilon)$ time units is p. For part (ii), the expected time for the CTMC to leave state s'_{iN_i} after entering z_{i1} is the sum of the mean of the Erlang-k_i distribution with rate λ_i^E and the mean of the exponential distributions with rates λ'_{i1} to λ'_{iN_i}, i.e.

$$k_i \tfrac{1}{\lambda_i^E} + \sum_{j=1}^{N_i} \tfrac{1}{\lambda'_{ij}} = \Delta_i + \sum_{j=1}^{N_i} \tfrac{1-\lambda_{ij}\delta_{ij}}{\lambda_{ij}} = \\ = \sum_{j=1}^{N_i} \delta_{ij} + \sum_{j=1}^{N_i} \left(\tfrac{1}{\lambda_{ij}} - \delta_{ij}\right) = \sum_{j=1}^{N_i} \tfrac{1}{\lambda_{ij}}.$$

□

Theorem 4 supports the calculation of the delay model parameters for a "together" state set $S_i = \{s_{i1}, s_{i2}, \ldots, s_{iN_i}\}$ as follows:

1) Approximate the delays $\delta_{i1}, \delta_{i2}, \ldots, \delta_{iN_i}$ for the components associated with each state from S_i using (10).
2) Compute the joint delay $\Delta_i = \sum_{j=1}^{N_i} \delta_{ij}$.
3) Choose a small error $\epsilon \in (0,1)$ and a small probability p (e.g. $\epsilon = 0.1$ and $p = 0.05$), and solve (17a) for k_i. This can be done using a numeric solver and rounding the result up to an integer value or, since k_i only depends on ϵ and p, and is independent of Δ_i, by using precomputed k_i values as in Table 4;
4) Calculate λ_i^E and λ'_{i1} to λ'_{iN_i} using (17b) and (17c), respectively.

Example 5. Consider the "together" set $S_1 = \{s_2, s_4\}$ for property **P1** from our running example (cf. Table 2). States s_2 and s_4 correspond to the invocations of the arrivals and search web services from the travel web application, which according to our experimental data have delays $\delta_2 = 45$ms and $\delta_4 = 209$ms, respectively. Therefore, the joint delay is $\Delta_1 = \delta_2 + \delta_4 = 254$ms. Suppose that we want to model this joint delay with an error bound $\epsilon = 0.1$ and a probability $p = 0.05$. This gives $k_1 = 259$ (cf. Table 4) and the other joint delay modelling parameters are calculated as: $\lambda_1^E = \frac{k_1}{\Delta_1} = \frac{259}{0.254} = 1019\text{s}^{-1}$, $\lambda'_2 = \frac{\lambda_2}{1-\lambda_2\delta_2} = \frac{19.88}{1-19.88\cdot 0.045} = 188.61\text{s}^{-1}$ and $\lambda'_4 = \frac{\lambda_4}{1-\lambda_4\delta_4} = \frac{1.85}{1-1.85\cdot 0.209} = 3.01\text{s}^{-1}$ (where the rates λ_2 and λ_4 are taken from Table 1).

The next theorem gives the format of the refined CTMC after joint delay modelling is applied to all "together" state sets S_1 to S_m.

Theorem 5. Applying the OMNI joint delay modelling procedure to the "together" state set S_i of a high-level model $\mathsf{CTMC}(S, \boldsymbol{\pi}, \mathbf{R})$ yields a model $\mathsf{CTMC}(S', \boldsymbol{\pi}', \mathbf{R}')$ with:

$$S' = (S \setminus S_i) \cup \{z_{i1}, z_{i2}, \ldots, z_{ik_i}, s'_{i1}, s'_{i2}, \ldots, s'_{iN_i}\};$$

$$\boldsymbol{\pi}'(s) = \begin{cases} \boldsymbol{\pi}(s), & \text{if } s \in S \setminus S_i \\ \boldsymbol{\pi}(s_{i1}), & \text{if } s = z_{i1} \\ 0, & \text{otherwise} \end{cases};$$

$$\mathbf{R}'(s,u) = \begin{cases} \mathbf{R}(s,u), & \text{if } s,u \in S \setminus S_i \\ \mathbf{R}(s,s_{i1}), & \text{if } s \in S \setminus S_i \wedge u = z_{i1} \\ \lambda_i^E, & \text{if } (s,u) \in \{(z_{i1},z_{i2}), \ldots, \\ & \quad (z_{i,k_i-1}, z_{ik_i}), (z_{ik_i}, s'_{i1})\} \\ \lambda'_{ij}, & \text{if } s = s'_{ij} \wedge u = s'_{i,j+1}, \\ & \quad 1 \leq j \leq N_i - 1 \\ \frac{\mathbf{R}(s_{iN_i},u)}{\lambda_{iN_i}} \lambda'_{iN_i}, & \text{if } s = s'_{iN_i} \wedge u \in S \setminus S_i \\ \frac{\mathbf{R}(s_{iN_i},s_{i1})}{\lambda_{iN_i}} \lambda'_{iN_i}, & \text{if } s = s'_{iN_i} \wedge u = z_{i1} \\ 0, & \text{otherwise} \end{cases}$$

if $s \neq u$; and $\mathbf{R}'(s,s) = -\sum_{u \in S' \setminus \{s\}} \mathbf{R}'(s,u)$, where the terms $\frac{\mathbf{R}(s_{iN_i},u)}{\lambda_{iN_i}}$ and $\frac{\mathbf{R}(s_{iN_i},s_{i1})}{\lambda_{iN_i}}$ correspond to the probabilities $p_1, p_2, \ldots$ from Fig. 6 and are obtained using (2).

Proof. The proof is by construction, cf. Fig. 6. □

4.3.2 Holding-time modelling

As indicated in Table 3, we model the holding times of system components associated with high-level CTMC states from $S_{\mathsf{O}} \cup S_1 \cup S_2 \ldots \cup S_m$ individually. For each such component, we synthesise a phase-type distribution $\mathsf{PHD}(\boldsymbol{\pi}_0, \mathbf{D}_0)$ that models the holding times (11), and we replace the relevant state s' of the model $\mathsf{CTMC}(S', \boldsymbol{\pi}', \mathbf{R}')$ obtained after the OMNI joint delay modelling with this PHD. For operations corresponding to states $s_{\mathsf{O}} \in S_{\mathsf{O}}$ the replaced state is $s' = s_{\mathsf{O}}$, while for operations corresponding to a

Algorithm 2 Holding-time modelling with parameters:
- $MinC$ — minimum number of PHD clusters
- $MaxC$ — maximum number of PHD clusters
- $MaxP$ — maximum number of cluster phases
- $FittingAlg$ — basic PHD fitting algorithm
- $MaxSteps$ — maximum steps without improvement

1: **function** HOLDINGTIMEMODELING($\alpha, \tau'_{i1}, \tau'_{i2}, \ldots, \tau'_{in_i}$)
2: $sample \leftarrow (\tau'_{i1}, \tau'_{i2}, \ldots, \tau'_{in_i})$
3: $minErr = \infty$
4: $improvement \leftarrow 0$
5: $steps \leftarrow 0$
6: $c \leftarrow MinC$
7: **while** $c \leq MaxC \wedge steps \leq MaxSteps$ **do**
8: $phd \leftarrow$ CBFITTING($sample, c, FittingAlg, MaxP$)
9: $err \leftarrow \Delta\mathsf{CDF}(sample, phd)$
10: **if** $err < minErr$ **then**
11: $best\_phd \leftarrow phd$
12: $improvement \leftarrow improvement + (minErr - err)$
13: $minErr \leftarrow err$
14: **end if**
15: **if** $improvement \geq \alpha$ **then**
16: $improvement \leftarrow 0$
17: $steps \leftarrow 0$
18: **else**
19: $steps \leftarrow steps + 1$
20: **end if**
21: $c \leftarrow c + 1$
22: **end while**
23: **return** $best\_phd$
24: **end function**

state $s_{ij} \in S_i$, $1 \leq i \leq m, 1 \leq j \leq N_i$, the replaced state is the state $s' = s'_{ij}$ obtained after the joint delay modelling of S_i (cf. Fig. 6b).

Our holding-time modelling exploits recent advances in the fitting of phase-type distributions to empirical data. Given the usefulness of PHDs in performance engineering, this area has received considerable attention [21], [55], with effective PHD fitting algorithms developed based on techniques such as moment matching [56], [57], expectation maximisation [58], [59], [60] and Bayes estimation [61], [62]. Recently, these algorithms have been used within PHD fitting approaches that: (a) partition the dataset into segments [60] or clusters [53] of "similar" data points; (b) employ an established algorithm to fit a PHD with a simple structure to each data segment or cluster; and (c) use these simple PHDs as the branches of a PHD that fits the whole dataset. These approaches achieve better trade-offs between the size, accuracy and complexity of the final PHD than the direct algorithms applied to the entire dataset.

The OMNI HOLDINGTIMEMODELING function from Algorithm 2 achieves similar benefits by employing Reinecke et al.'s *cluster-based PHD fitting* approach [53], [63], [64] to fit a PHD to the holding time sample $\tau'_{i1}, \tau'_{i2}, \ldots, \tau'_{in_i}$ from (11). The PHD fitting is carried out by the while loop in lines 7–22, which iteratively assesses the suitability of PHDs obtained when partitioning the $sample$ assembled in line 2 into $c = MinC, MinC + 1, \ldots, MaxC$ clusters. Line 8 obtains a PHD with c branches (corresponding to partitioning $sample$ into c clusters) and up to $MaxP$ phases by using the function CBFITTING, which implements the cluster-based PHD fitting from [53]. The $FittingAlg$ argument of CBFITTING specifies the basic PHD fitting algorithm applied to each cluster as explained above, and can be any of the standard moment matching, expectation maximisation or Bayes estimation PHD fitting algorithms. The quality of the c-branch PHD is assessed in line 9 by using the CDF-difference metric [53] to compute the difference err between $sample$ and the PHD. The if statement in lines 10–14 identifies the PHD with the lowest err value so far, retaining it in line 11. Reductions in err (i.e., "improvements") are cumulated in $improvement$ (line 12), and the while loop terminates early if the iteration counter $steps$ exceeds $MaxSteps$ before $improvement$ reaches the threshold $\alpha \geq 0$ provided as an parameter to HOLDINGTIMEMODELING and the $steps$ counter is reset in line 17. Finally, the best PHD achieved within the while loop is returned in line 23.

Theorem 6. Using a phase-type distribution $\mathsf{PHD}(\boldsymbol{\pi}_0, \mathbf{D}_0)$ generated by Algorithm 2 to apply the OMNI holding-time modelling procedure to the state s' of a model $\mathsf{CTMC}(S', \boldsymbol{\pi}', \mathbf{R}')$ yields a model $\mathsf{CTMC}(S'', \boldsymbol{\pi}'', \mathbf{R}'')$ with:

$$S'' = (S' \setminus \{s'\}) \cup \{w_1, w_2, \ldots, w_N\} ;$$

$$\boldsymbol{\pi}''(s) = \begin{cases} \boldsymbol{\pi}'(s), & \text{if } s \in S \setminus \{s'\} \\ \boldsymbol{\pi}'(s')\boldsymbol{\pi}_0(w_i), & \text{if } s = w_i, 1 \leq i \leq N \\ 0, & \text{otherwise} \end{cases} ;$$

$$\mathbf{R}''(s,u) = \begin{cases} \mathbf{R}'(s,u), & \text{if } s,u \in S' \setminus \{s'\} \\ \mathbf{D}_0(s,u), & \text{if } s,u \in \{w_1, w_2, \ldots, w_N\} \\ \mathbf{R}'(s,s')\boldsymbol{\pi}_0(w_i), & \text{if } s \in S' \setminus \{s'\} \wedge u = w_i, \\ & \quad 1 \leq i \leq N \\ \frac{\mathbf{R}'(s',u)}{\lambda'}\mathbf{d}_1(w_i), & \text{if } s = w_i \wedge u \in S' \setminus \{s'\}, \\ & \quad 1 \leq i \leq N \end{cases}$$

if $s \neq u$; and $\mathbf{R}''(s,s) = -\sum_{u \in S'' \setminus \{s\}} \mathbf{R}''(s,u)$, where:

- $w_1, w_2, \ldots, w_N$ are the transient states of $\mathsf{PHD}(\boldsymbol{\pi}_0, \mathbf{D}_0)$;
- λ' is the total outgoing transition rate for s';
- $\mathbf{d_1} = -\mathbf{D}_0\mathbf{1}$.

Proof. The proof is by construction, cf. Algorithm 2. □

Example 6. We used our OMNI refinement tool (Section 5) to perform the component classification and selective refinement steps of our approach on the high-level CTMC from the motivating example. Algorithm 2 was executed for each component associated with a CTMC state from the S_0 or the S_1 to S_m state sets in Table 2, with $\alpha = 0.1$ and with the configuration parameters $MinC = 2$, $MaxC = 30$, $MaxP = 300$, $MaxSteps = 3$ and $FittingAlg$ an expectation-maximisation PHD fitting algorithm that produces hyper-Erlang distributions.[4] We obtained refined CTMCs comprising 730 states and 761 transitions for property **P1**, 367 states and 387 transitions for property **P2**, and 730 states and 761 transitions for property **P3**. Fig. 7 compares the actual values of properties **P1**–**P3** with the values predicted by the

4. A hyper-Erlang distribution [21], [60], [65] is a PHD in which the $c > 1$ branches of the PHD from Algorithm 2 are mutually independent Erlang distributions.

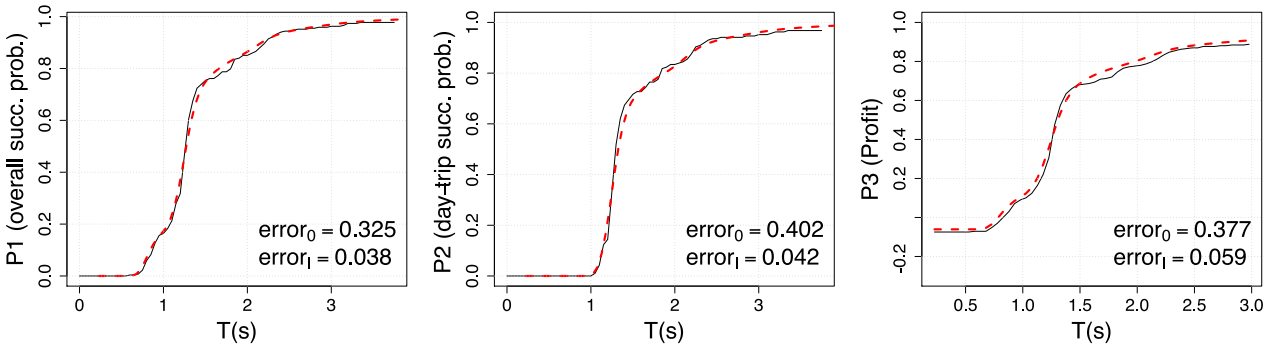


Fig. 7: Actual (continuous lines) property values versus property values predicted (dashed lines) using the OMNI-refined CTMC model; error_0 and error_I represent the error (9) for the high-level CTMC and the refined CTMC, respectively.

analysis of these refined CTMCs. Both the visual assessment and the error values error_I associated with these predictions (which are significantly lower than the error values error_0 before refinement) show that OMNI supports the accurate analysis of the three properties. In fact, the predicted and actual values for all properties may seem surprisingly close. The explanation for this close match is twofold. First, the OMNI refinement uses PHD distributions, which – if sufficiently large – can approximate arbitrarily close any continuous distribution (cf. Section 2.3). Second, the apparently "perfect" match between the predicted and the actual QoS property values is slightly deceptive: for instance, a closer inspection of the results shows that there are still multiple points where the difference between the two is at least 5%. In Section 6.2 we supplement this brief discussion of the results from Fig. 7 with an experimental evaluation which shows that the accurate OMNI results are not due to overfitting.

5 OMNI Refinement Tool

We implemented OMNI as a Java tool that takes as input a high-level CTMC model. This model is specified in a variant of the PRISM modelling language [14] where state transition commands are expressed using components labels. For example, the PRISM command

$$\mathsf{s}=1 \;\rightarrow\; \mathsf{p}_1 * \lambda_1 : (\mathsf{s}'=2) + (1-\mathsf{p}_1) * \lambda_1 : (\mathsf{s}'=3);$$

that defines the outgoing transitions for state s_1 of our high-level CTMC model from Fig. 2 is replaced by

$$\mathsf{s}=\langle location\rangle \;\rightarrow\; \mathsf{p}_1 : (\mathsf{s}'=\langle arrival\rangle) + (1-\mathsf{p}_1) : (\mathsf{s}'=\langle departures\rangle);$$

in the OMNI variant of the modelling language. This indicates that the CTMC transitions from the state associated with the *location* component to either the state associated with the *arrival* component (with probability p_1) or to the state associated with the *departures* component (with probability $1-p_1$). An XML configuration file is then used to map each of these component labels to a file of comma-separated values containing the observed execution times for the relevant component. In addition, this configuration file allows the user to define the OMNI refinement parameters (i.e. k_i from Theorem 4, and α, $MinC$, $MaxC$, $MaxP$ and $MaxSteps$ from Algorithm 2). The $FittingAlg$ parameter is fixed in the current version of the tool, so that OMNI uses the expectation-maximisation PHD fitting algorithm mentioned in Example 6.

When multiple QoS properties (for the same high-level CTMC) are provided to the OMNI tool, we avoid the overheads associated with the repeated execution of the modelling tasks from Table 3 for the same components by maintaining a cache of all completed tasks and their results. As such, each of these tasks is executed at most once per system component, and its cached result is used when needed instead of repeating the task. By comparison, refining the whole CTMC indiscriminately for even a single QoS property would require the execution of these modelling tasks for every system component.

Finally, to support the scenario where the component delays (10) are negligible compared to the holding times (11), the configuration file allows the specification of a *delay threshold*, and components with delays (10) below this threshold are not included in the joint delay modelling step of the OMNI refinement. We found experimentally that this leads to significant reductions in the size of the refined CTMC with no impact on the accuracy of the QoS analysis.

Our OMNI tool uses the HyperStar PHD fitting tool from [63] for the CBFitting function from Algorithm 2, and produces the refined CTMCs as standard PRISM models. The OMNI tool is freely available from our project webpage https://www.cs.york.ac.uk/tasp/OMNI/, together with detailed instructions and all the models and datasets from this paper.

6 Evaluation

We evaluated OMNI by performing a set of experiments aimed at answering the following research questions.

RQ1 (Accuracy/No overfitting): How effective are OMNI models at predicting QoS property values for other system runs than the one used to collect the execution-time observation datasets for the refinement?

RQ2 (Refinement granularity): What is the effect of varying the OMNI refinement granularity on the refined model accuracy, size and verification time?

RQ3 (Training dataset size): What is the effect of the training dataset size on the refined model accuracy?

RQ4 (Component classification): What is the benefit of using a component classification step within OMNI?

To assess the generality of OMNI, we carried out our

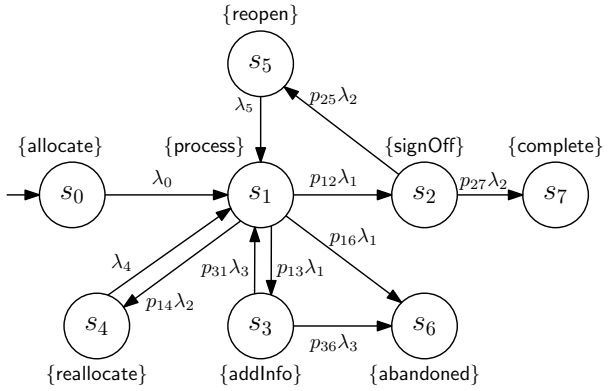


Fig. 8: High-level CTMC model of IT support system

experiments within two case studies that used real systems and datasets from different application domains. The first case study is based on the travel web application presented in Section 3 and used as a motivating example earlier in the paper. In the second case study, we applied OMNI to an IT support system. This system is introduced in Section 6.1, followed by descriptions of the experiments carried out to address the four research questions in Sections 6.2–6.5.

6.1 IT Support System

The real-world IT support system we used to evaluate OMNI is deployed at the Federal Institute of Education, Science and Technology of Rio Grande de Norte (IFRN), Brazil. The system enables the IFRN IT support team to handle user tickets reporting problems with the institute's computing systems. As part of our collaboration with IFRN researchers [48], system logs covering the handling of 1410 user tickets were collected from this IT support system over a period of six months between September 2016 and February 2017.

A high-level CTMC model of the business process implemented by the IT system is shown in Fig. 8. In this model, state s_0 corresponds to a ticket being created by a "client" and awaiting allocation to a member of the support team. Once allocated, the ticket is processed (state s_1) and, if the issue can be resolved, the client is informed and the ticket awaits sign off (s_2) before being marked as complete (s_7). The client may choose to reopen (s_5) the ticket rather than close it, in which case the ticket is returned to the support team member for further processing. Whilst processing a ticket, the support staff may require additional information from the client (s_3) or may need to reallocate the ticket to another member of the IT support team (s_4). A ticket may also be abandoned (s_6) either during processing or whilst awaiting additional information from the client.

We used our OMNI tool to refine the high-level CTMC from Fig. 8 in order to support the verification of the response time of the IT support system through the analysis of two properties:

$$\begin{array}{ll} \textbf{P1} & P_{=?}[F^{[0,T]}\, complete] \\ \textbf{P2} & P_{=?}[(\neg reopen\ \&\ \neg addInfo)\ U^{[0,T]}\, complete] \end{array} \quad (18)$$

where **P1** specifies the probability of a ticket reaching the complete state within T (working) hours, and **P2** represents the probability of ticket handling being completed within T working hours without further input from the client who raised the ticket and without the ticket being reopened.

TABLE 5: Execution rates for the IT support system

Component	Rate (hours^{-1})
allocate	$\lambda_0 = 0.08248$
process	$\lambda_1 = 0.09799$
signOff	$\lambda_2 = 0.01167$
addInfo	$\lambda_3 = 0.02006$
reallocate	$\lambda_4 = 0.02839$
reopen	$\lambda_5 = 0.09988$

TABLE 6: Transition probabilities for the IT support system

CTMC states		Transitions from s_i to s_j	Transitions leaving s_i	Estimate transition probability $s_i \to s_j$
s_i	s_j			
s_1	s_2	533	705	$p_{12} = 533/705 = 0.76$
s_1	s_3	24	705	$p_{13} = 24/705 = 0.03$
s_1	s_4	34	705	$p_{14} = 34/705 = 0.05$
s_1	s_6	114	705	$p_{16} = 114/705 = 0.16$
s_2	s_5	501	533	$p_{25} = 501/533 = 0.94$
s_2	s_7	32	533	$p_{27} = 32/533 = 0.06$
s_3	s_1	16	24	$p_{31} = 16/24 = 0.67$
s_3	s_6	8	24	$p_{36} = 8/24 = 0.33$

We used only half of the six-month logs (covering 705 tickets created over the first approximately three months) for the OMNI refinement, so that we could use the other half of the logs to answer research question RQ1 (cf. Section 6.2.2). For each ticket, the time spent in a particular state was derived from the log entries, taking into account only the working hours for the IT support team.[5] Assuming exponentially distributed execution times for the components of the IT support process, we used (6) to calculate the component execution rates shown in Table 5. Finally, we used the logs to calculate the frequencies of state transitions, and thus to estimate the CTMC state transition probabilities as shown in Table 6.

Fig. 9 compares the actual values of properties **P1** and **P2** from (18) – computed based on the system logs – with the values predicted by the analyses of: (a) the high-level CTMC from Fig. 8 (with the parameters given in Tables 5 and 6); and (b) OMNI-refined CTMC models for the two properties. The refined CTMCs were obtained using the same OMNI parameters as in Example 6, except $\alpha = 0.2$ and a *delay threshold* of 0.01 hours.[6] As explained in Section 5, this threshold meant that components with a delay (10) below 0.01 hours (which amounted to all component of the IT support system) were not included in the joint delay modelling of OMNI.

Having introduced the system used in our second case study, we will use the next sections to describe the experiments carried out to answer our four research questions.

6.2 RQ1 (Accuracy/No overfitting)

The generation of OMNI-refined CTMC models requires the processing of finite datasets produced by the components of

5. The working hours for the period covered by the logs were identified through consultation with the IFRN owner of the IT support process.

6. The threshold value was chosen to be approximately three orders of magnitude smaller than the smallest mean execution time of a system component, i.e. $1/\lambda_5 = 10.012$ hours for the IT support system (cf. Table 5).

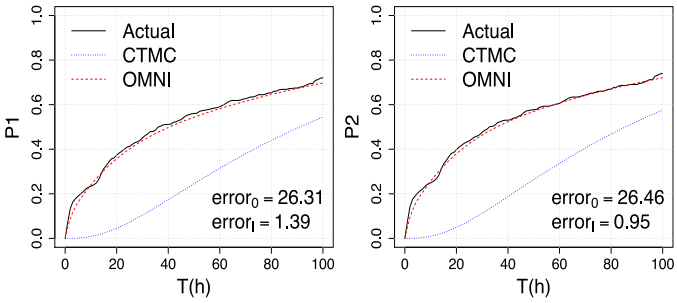


Fig. 9: Actual values of the IT support system properties versus property values predicted using the high-level and the refined CTMC models, over 100 working hours from ticket creation; the prediction error (9) for the refined CTMCs (i.e. $error_I$) is 94.7% smaller (for property **P1**) and 97% smaller (for property **P2**) than the corresponding prediction errors for the high-level CTMC (i.e. $error_0$).

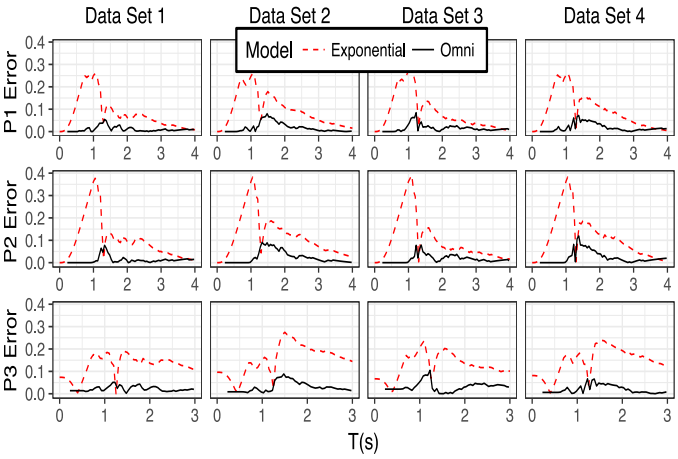


Fig. 10: Prediction error for the web application properties, for training and testing datasets from different runs

the analysed system, in order to extract key model features. To be useful, these CTMCs should accurately predict the values of the system properties for other system runs, i.e. should not be overfitted to the datasets used to generate them.

6.2.1 Travel Web Application

To assess whether the OMNI web application models possess this property, we obtained three additional datasets (labelled 'Data Set 2', 'Data Set 3' and 'Data Set 4') for the travel web application. Each new dataset corresponds to a four-hour run, with all datasets (including the original dataset, 'Data Set 1') captured over a period of two days. Fig. 10 shows the difference between the property values predicted by the CTMC analysis and the actual property values taken from each of the four datasets. Results are shown for the initial CTMC from Fig. 2 (labelled 'Exponential' in the diagrams) and the refined models obtained using 'Data Set 1' (labelled 'Omni' in the diagrams). In all cases, the OMNI-refined CTMCs significantly improve the accuracy of the analysis when compared to the traditional CTMC analysis approach.

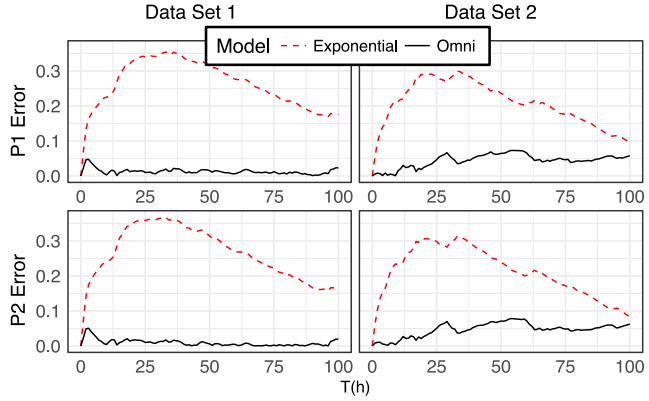


Fig. 11: Prediction error for the IT support system properties, for training and testing datasets from different three-month time periods

6.2.2 IT Support System

For the IT support system, OMNI-refined CTMCs for the two properties were produced from half of the available system logs ('Data Set 1') as described in Section 6.1. We then assessed the accuracy of the predictions obtained using these refined CTMCs against the actual property values extracted from the training 'Data Set 1' and from the test dataset ('Data Set 2') produced from the second half of the system logs. Fig. 11 shows the error when the predicted values are compared to actual values for the two datasets. For both datasets, the OMNI-refined CTMCs produce results which significantly outperform the results obtained by analysing the high-level CTMC.

6.2.3 Discussion

The experiments described in the previous sections show that OMNI consistently outperformed the traditional CTMC modelling and analysis approach in both case studies, irrespective of the choice of training set. To confirm that OMNI delivers error reductions reliably, we performed additional experiments in which the training and testing datasets were drawn randomly from all available observations. Thirty experiments were carried out for each of our two systems, and the results displayed similar error reductions to those presented in Sections 6.2.1 and 6.2.2. This shows that OMNI models can effectively predict QoS property values for other system runs than the one used to collect the training datasets employed in the refinement.

Our additional experiments also showed that the error profiles from Figs. 10 and 11 capture several general features for the type of QoS analysis improved by OMNI:

1. The initial peak in the 'Exponential' prediction error for properties **P1** and **P2** from Fig. 10 is characteristic of the inability of exponential distributions to model delays, as also explained in Section 4.1. OMNI does not suffer from this limitation. Note that this modelling error does not affect properties **P1** and **P2** from Fig. 11 because the delays for the IT support system are insignificant compared to the holding times.
2. The second peak in the 'Exponential' prediction error for properties **P1** and **P2** from Fig. 10, and the first peak for

properties **P1** and **P2** from Fig. 11 are representative of the inability of exponential distributions to model long tails (due to operations occasionally having much longer execution times than their typical execution times). As such, the estimated rates of the exponential distributions are too low, and the predictions are overly conservative. These error peaks are particularly high (above 0.3) for the IT support system, as IT support personnel occasionally required very long times to address a user request. Again, OMNI yields much smaller prediction errors around these peaks.

3. The multiple peaks in the 'Exponential' prediction error for property **P3** from Fig. 10 is characteristic of derived properties, i.e., properties defined using multiple "primitive" properties (in this case, **P3** represents profit, and is defined as the difference between revenue and penalties). The multiple peaks are due to the prediction errors for the primitive properties peaking at different time moments. As before, OMNI significantly dampens these peaks.

6.3 RQ2 (Refinement Granularity)

To evaluate the effects of refinement granularity we constructed a set of OMNI models by varying:

1) k_i, the number of states in the Erlang delay models from the joint delay modelling of OMNI (cf. Theorem 4);
2) α, the PHD model fitting threshold used in the holding time modelling of OMNI (cf. Algorithm 2).

Larger values of k_i are associated with increased accuracy in the modelling of delays, whilst reducing α corresponds to finer-grained refinement in the PHD modelling.

6.3.1 Travel Web Application

The experimental results from the web application case study are presented in Table 7. As k_i is increased from 10 to 100 and from 100 to 259, the error is reduced.[7] However, this improvement shows diminishing returns for all properties as k_i becomes large. The same pattern occurs as α is decreased, with smaller errors for smaller α values but only a marginal reduction in error as α is reduced from 0.1 to 0.05.

Since k_i controls the number of states associated with delays, increasing k_i also increases the total number of states associated with the model. The model size also increases as α is decreased.

Finally, the experimental results confirm that the models for property **P2** are consistently much smaller than for **P1** and **P3** since more states from the initial CTMC are in the "exclude from refinement" set S_{X} when evaluating **P2** than when evaluating the other properties (cf. Table 2). T_V is the total time for PRISM to verify each property in the interval [0, $T_{\max}$] with a time step of 0.05s and includes the time taken for model construction. All experiments presented here and throughout the rest of the paper were carried out on a MacBook Pro with 2.9 GHz Intel i5 processor and 16Gb of memory. As the model increases in size, and accuracy improves, the time taken for verification also increases, up to 29.4s for the finest-grained model used to evaluate property **P3** across the entire interval [0, $T_{\max}$].

7. These k_i values are taken from Table 4.

6.3.2 IT Support System

When OMNI is applied to the IT support system, the delay threshold of 0.01 hours chosen as explained in Section 6.1 means that the delay modelling was omitted (i.e. delays were approximated to zero). As such, we were not interested in varying k_i in this case study, and Table 8 only shows the effects of decreasing α on the refined models. Like in the first case study, decreasing α gradually reduces the prediction error, with a significant error reduction obtained even for the largest α from our experiments (e.g. an over tenfold reduction from 26.3 for the initial, high-level CTMC and property **P1** to just 2.45 for the coarsest-granularity CTMC generated for $\alpha = 0.6$). Diminishing returns in terms of error reduction are achieved for property **P2**; for **P1**, this trend is not clearly distinguishable for the tested α values.

As expected, the model size grows as α is decreased, leading to a corresponding increase in the verification time T_V. T_V includes the time for the construction of the model and for PRISM to analyse the property in the interval $[0, 100\text{h}]$ with a time step of one hour (i.e. 100 verification sessions). The largest verification time is 274.6s for the finest-granularity CTMC obtained for property **P1**, which is entirely acceptable for an offline verification task.

During the component classification step of OMNI, the exclusion sets for the two properties are calculated as $S_{\mathsf{X}} = \{s_6, s_7\}$ for **P1** and $S_{\mathsf{X}} = \{s_3, s_5, s_6, s_7\}$ for **P2**. Therefore, the models associated with **P2** are consistently smaller than those associated with **P1**, whose exclusion set S_{X} contains only two states.

6.3.3 Discussion

For both case studies and all considered QoS properties, considerable improvements in model accuracy are obtained even with small, coarse-grained OMNI models. As such, OMNI can offer significant improvements in accuracy over traditional CTMC modelling techniques even when computational resources are at a premium. Additional, but typically diminishing, gains in prediction accuracy are obtained through increasing the granularity of the refinement. Expectedly, this leads to a corresponding increase in verification time. For our two systems, this time did not exceed 10 minutes (and was typically much smaller) for all considered properties and model granularities – an acceptable overhead for the offline verification task performed by OMNI.

6.4 RQ3 (Training dataset size)

In both case studies, we ran a set of experiments to evaluate the effect of reducing the training dataset size on the accuracy of OMNI models. For each experiment, training subsets were constructed by randomly selecting a percentage of all available datasets used to answer the previous research questions. The sizes of these selected subsets were 80%, 60%, 40% and 20% of the complete training dataset from Sections 6.2.1 and 6.2.2. For each system and each of its analysed QoS properties, the experiments were repeated 30 times, with the property errors recorded.

TABLE 7: Effects of the OMNI refinement granularity on web application model

		P1			P2			P3		
k_i	α	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$
Initial CTMC		7	0.325	1.9	7	0.402	1.9	7	0.377	1.9
10	0.2	82	0.085	3.9	45	0.126	3.5	82	0.094	5.0
100	0.2	262	0.049	5.6	135	0.066	4.3	262	0.077	7.7
259	0.2	580	0.045	8.8	294	0.060	5.8	580	0.074	12.6
10	0.1	232	0.078	6	118	0.112	4.3	232	0.083	8.1
100	0.1	412	0.043	7.8	208	0.049	5.1	412	0.063	11.0
259	0.1	730	0.038	11.4	367	0.042	6.8	730	0.059	16.5
10	0.05	618	0.075	13.8	376	0.106	7.9	618	0.081	19.6
100	0.05	798	0.041	16.0	466	0.044	8.8	798	0.061	22.7
259	0.05	1116	0.036	20.6	625	0.036	10.8	1116	0.057	29.4

TABLE 8: Effects of the OMNI refinement granularity on the IT support system model

	P1			P2		
α	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$
Initial CTMC	8	26.3	3.0	8	26.5	3.0
0.6	44	2.45	42.8	41	2.11	41.7
0.4	61	2.24	48.2	51	1.74	44.4
0.2	202	1.39	95.8	174	0.95	84.4
0.1	329	1.27	139.6	259	0.87	110.6
0.05	722	1.01	274.6	346	0.84	139.8

TABLE 9: Web application – training dataset size effect on prediction accuracy, shown as average error and standard deviation over 30 runs

Dataset†	**P1 Error**	**P2 Error**	**P3 Error**
100%††	0.038 sd N/A*	0.042 sd N/A*	0.059 sd N/A*
80%	0.038 sd 0.006	0.043 sd 0.005	0.058 sd 0.023
60%	0.046 sd 0.013	0.048 sd 0.014	0.078 sd 0.041
40%	0.057 sd 0.017	0.063 sd 0.022	0.105 sd 0.054
20%	0.083 sd 0.032	0.076 sd 0.026	0.156 sd 0.075
Initial CTMC	0.325 sd N/A*	0.402 sd N/A*	0.377 sd N/A*

†Percentage of complete 270-element training dataset
††Single run using entire data set
*Single run, so no standard deviation

6.4.1 Travel Web Application

Table 9 shows the mean error and standard deviation (labelled 'sd') for the web application case study with $k_i = 259$ and $\alpha = 0.1$. As the training dataset size decreases, the error and standard deviation associated with each property show an increasing trend. However, we note that at 80% the prediction errors show little difference to the 100% figures – the mean errors at 100% are very close to the 80% errors, and well within one standard deviation of the 80% mean. This suggests that 80% of the complete dataset is sufficient to capture the characteristics of the underlying component distributions for this case study.

6.4.2 IT Support System

For the IT support system, the experimental results are provided in Table 10. As for the other case study, we observe that reducing the size of the training sets leads to a trend where the prediction error and the standard deviation increases. This also happens when the size of the training dataset is reduced from 100% to 80%, suggesting that additional slight improvements may be possible by further increasing the size of the initial training dataset.

TABLE 10: IT support system – training dataset size effect on prediction accuracy, shown as average error and standard deviation over 30 runs

Dataset†	**P1 Error**	**P2 Error**
100%††	1.39 sd N/A*	0.95 sd N/A*
80%	1.53 sd 0.774	1.35 sd 0.570
60%	1.57 sd 0.942	1.55 sd 0.797
40%	2.37 sd 1.450	2.19 sd 1.396
20%	3.70 sd 2.825	3.81 sd 3.039
Initial CTMC	26.31 sd N/A*	26.46 sd N/A*

†Percentage of complete 705-element training dataset
††Single run using entire data set
*Single run, so no standard deviation

6.4.3 Discussion

For both case studies we note that even modest training dataset sizes show a significant improvement over the traditional approach to CTMC-based analysis of QoS properties. For the web application, a training set consisting of 20% of the original dataset equates to only 54 request handling observations, and reduces the mean estimation error by between 50–81% for the properties of interest. For the IT system, 20% of the original training dataset equates to 141 tickets processed, with the processing of only five tickets using the addInfo component of the system, yet the prediction errors for **P1** and **P2** are both reduced by approximately 86%.

6.5 RQ4 (Component classification)

We evaluated the effects of extending our preliminary CTMC-refinement method from [31] with the component classification step described in Section 4.2. To this end, we performed experiments to compare the model size, verification time and accuracy of the refined CTMCs generated by the OMNI method described in this paper and of the refined CTMCs produced by our preliminary method, which refines each system component independently, irrespective of whether it impacts the analysed QoS property or not.

TABLE 11: Web application – comparison of OMNI with the preliminary CTMC refinement approach from [31]

	P1			P2			P3		
Model	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$
Preliminary OMNI ($\alpha = 0.1$, $k_i = 259$)	1766	0.037	24.95	1766	0.039	23.79	1766	0.063	38.21
OMNI ($\alpha = 0.1$, $k_i = 259$)	730	0.038	11.40	367	0.042	6.80	730	0.059	16.5
OMNI ($\alpha = 0.05$, $k_i = 259$)	1116	0.036	20.6	625	0.036	10.8	1116	0.057	29.4
High-level CTMC	7	0.325	2.53	7	0.402	2.50	7	0.377	2.47

TABLE 12: IT support system – comparison of OMNI with the preliminary CTMC refinement approach from [31]

	P1			P2		
Model	**#states**	**Error**	$T_V(s)$	**#states**	**Error**	$T_V(s)$
Preliminary OMNI ($\alpha = 0.2$, $k_i = 10$)	265	1.39	261.5	265	0.95	239.6
OMNI ($\alpha = 0.2$)	202	1.39	95.8	174	0.95	84.4
High level CTMC	8	26.3	3.0	8	26.5	3.0

6.5.1 Travel Web Application

For the web application, refined CTMCs were built using first the preliminary OMNI method from [31] and the fully fledged version of OMNI from this paper, initially with parameters $k_i = 259$ and $\alpha = 0.1$. The first two rows from Table 11 summarise these experimental results, which show that the use of component classification yields significant reductions in the number of model states and the verification time for all three properties compared to the preliminary method. As expected given the CTMC state partition from Table 2 and OMNI rules from Table 3, the largest reductions are achieved for property **P2** (79% fewer model states, and 71% shorter verification time). For properties **P1** and **P3**, the use of component classification led to a reduction in model size of over 58%, and to reductions in verification time of 54% and 57%, respectively.

The prediction errors are very close for both OMNI variants, and considerably smaller than the errors for the high-level CTMC (provided in the last row of Table 11 for convenience). However, the errors are negligibly larger when component classification is used. This is due to the use of fewer states for OMNI's joint delay modelling compared to the separate modelling of component delays in [31]. As shown in Table 7, additional reductions in prediction error may be achieved by increasing k_i or reducing α if required. For example, by setting $\alpha = 0.05$, the refined models still have much fewer states than the preliminary OMNI model, show an improvement in verification time of between 17–54%, and are more accurate. The third row of Table 11 shows again these experimental results for ease of comparison.

6.5.2 IT Support System

For the second case study, the experimental results are presented in Table 12. Whilst our fully fledged OMNI allows for component delays to be omitted from the refinement when they are below a delay threshold (cf. Section 5), this was not possible in our previous work. Therefore, to ensure a fair comparison, we used a small k_i value ($k_i = 10$) when generating refined CTMCs with the preliminary OMNI variant. As shown by the experimental results, using the fully fledged OMNI yields smaller refined models that take 74% and 64.7% less time to verify for the IT system properties **P1** and **P2**, respectively. Furthermore, these smaller refined models achieve the same prediction accuracy as the larger models generated by the preliminary OMNI.

6.5.3 Discussion

The OMNI method described in this paper includes a component classification step in its model construction. Since this step uses the high-level CTMC model only, the time taken for its execution is very small. For the web application, the time taken to classify the high-level CTMC states for all three properties was 2.3s, and for the IT support system this step took only 1.8s.

For both case studies presented we have shown that for all the properties considered it was possible to generate OMNI models which are smaller, faster to verify and no less accurate than those produced by our preliminary approach from [31]. The amount of verification time saved depends on the number of states for which delays can be combined, and on the number of states which can be excluded from refinement – but these savings were considerable in all our experiments with the two real-world systems.

7 Threats to Validity

7.1 External validity

External validity threats may arise if the stochastic characteristics of the systems from our case studies are not indicative of the characteristics of other systems. To mitigate this threat, we used two significantly different systems from different domains for the OMNI evaluation. The section labelled 'System' from Table 13 summarises the multiple characteristics that differ between these systems.

In addition, the datasets used in the two case studies present different characteristics, as shown in the 'Datasets' section from Table 13. In particular, the datasets for the service-based system were obtained from real web services, while for the IT support system they were taken from the actual system logs. This gives us confidence that the stochastic characteristics of the two systems (including regions of

TABLE 13: Characteristics of the case studies used to evaluate OMNI

Characteristic	Case study 1	Case study 2
System		
Type of system	Prototype service-based system developed by the OMNI team	Production IT support system developed by, running at, and managed by university in Brazil
System components	Mix of six high-performance (commercial) and budget (free) third-party web services invoked remotely over the Internet	Proprietary software components deployed on university computing infrastructure, and supporting human tasks with high variance in temporal characteristics
Size of system	3188 lines of code	1276131 lines of code
Component execution times	Tens to hundreds of milliseconds	Minutes to hours
Operational profile	Assumed values for the probabilities of the different types of requests	Probabilities of different operation outcomes extracted from the real system logs
Datasets		
Dataset source	Obtained from invocations of real web services	Taken from actual system logs
Key dataset features	Significant delays (compared to holding times) due to network latency	Long tails and outliers due to a small number of complex user tickets; multi-modal response times and regions of zero density due to different experience levels of IT support personnel; negligible delays
Analysed QoS properties		
Types of properties	Overall success probability (**P1**) Success probability for "day-trip" requests (**P2**) Profit = revenue − penalties (**P3**)	Overall response time (**P1**) Response time for "straightforward" user tickets (**P2**)
Purpose of QoS analysis		
Supported stage of development process	Design of new system	Verification of existing system

zero density, multi-modal response times, and long tails) are representative for many real-world systems.

The next section from Table 13 summarises the different types of QoS properties analysed in our case studies. The transient fragment of continuous stochastic logic (whose analysis accuracy is improved by OMNI supports, cf. Section 2.2) supports the specification of multiple classes of QoS properties of interest, including success probability, profit/cost and response time, and our case studies considered examples of all of these.

Finally, as shown in the 'Purpose of QoS analysis' section from Table 13, the first case study applied OMNI during the design stage of the development process, whereas the second case study assessed OMNI by verifying QoS properties of an existing system.

Another external threat may arise if the OMNI-refined CTMC models were too large to be verified within a reasonable amount of time. The OMNI approach mitigates this threat by allowing for the refinement to be carried out at different levels of granularity, and our experiments indicate that significant improvements in prediction accuracy is achievable with modest enlargement of the models.

Our two case studies are based on real systems, but these systems have a relatively small number of *modelled components*. For the large IT system from the second case study, the small number of *modelled components* is due to the inclusion in the model of only components that influence the analysed QoS properties – a modelling technique termed *abstraction*.

For systems with larger numbers of modelled components, we note that the increase in model size due to the OMNI refinement is only linear in the number of system components. Moreover, as OMNI uses acyclic PHDs, the number of transitions also increases linearly. Modern model checkers can handle CTMCs with $10^5 - 10^6$ states [66] and as such we expect OMNI to scale well with much larger systems. We confirmed these hypotheses by constructing models with 12, 24, 48 and 96 components by combining 2, 4, 8 and 16 instances of our web application CTMC from Fig. 2.[8] OMNI was then used to refine the composite models with $k_i = 259$ and $\alpha = 0.1$. For each refined CTMC, we measured the number of states, the number of transitions, and the time taken to verify property **P1** of the travel web application at the single time point $T = 20s$ (since **P2** and **P3** can not be meaningfully extrapolated to these larger systems). The results of these experiments, shown in Fig. 12, confirm the predicted linear increase in the verification overhead with the system size.

7.2 Construct validity

Construct validity threats may be due to the assumptions made when collecting the datasets or when defining the QoS properties for our model refinement experiments. To address the first threat, we collected the datasets from a real IT support system and from a prototype web application that we implemented using standard Java technologies and six real web services from three different providers. For the first system, the datasets were collected over a period of six months, and for the second system they were collected on two different days, at different times of day. Furthermore, we used different datasets for training and testing. To

8. We did not perform similar experiments for the IT support system as they would not have been qualitatively different.

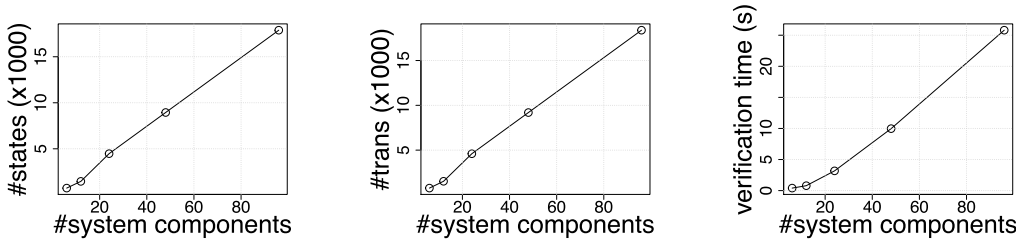


Fig. 12: Refined CTMC states, transitions and verification time for property **P1** at a single time point, for system sizes up to 16 times larger than the web application

mitigate the second threat, we analysed three performance and cost properties of web application, and two typical performance properties of the IT system.

7.3 Internal validity

Internal validity threats can originate from the stochastic nature of the two analysed systems or from bias in our interpretation of the experimental results. We addressed these threats by provided formal proofs for our CTMC refinement method, by reporting results from multiple independent experiments performed for different values of the OMNI parameters, and by analysing several QoS properties at multiple levels of refinement granularity. Additionally, we made the experimental data and results publicly available on our project webpage in order to enable the replication of our results.

8 Related work

To the best of our knowledge, OMNI is the first tool-supported method for refining high-level CTMC models of component-based systems based on separate observations of the execution times of the system components.

OMNI builds on recent approaches to using PHDs to fit non-parametric distributions, a research area that has produced many efficient PHD fitting algorithms over the past decade [57], [60], [61], [62]. Buchholz et al. [21] and Okamura and Dohi [55] present overviews of the theory and applications of PHDs in these types of analysis, in domains including the modelling of call centres [44] and healthcare processes [43], [45]. However, these algorithms and applications consider the distribution of timing data for a complete end-to-end process rather than separate timing datasets for the components of a larger system as is the case for OMNI. This focus on a single dataset also applies to the cluster-based PHD fitting method from [53] and its implementation within the efficient PHD-fitting tool HyperStar [63], which OMNI uses for its holding-time modelling.

Recent work by Karmakar and Gopinath [67] has shown that PHD models can be used in conjunction with CTMC solvers to verify storage reliability models. In this work, Weibull distributions are assumed to more accurately describe the processes of concern, and PHDs are used to approximate these distributions. The PRISM probabilistic model checker is then used to assess properties concerned with system reliability. Unlike this approach, OMNI is applicable to the much wider class of problems where additional QoS properties need to be analysed and where the relevant component features correspond to non-parametric distributions that cannot be accurately modelled as Weibull distributions.

The analysis of non-Markovian processes using PHDs is considered in [68], where a process algebra is proposed for use with the probabilistic model checker PRISM. However, [68] presents only the analysis of a simple system based on well-known distributions, and does not consider PHD fitting to real data nor how its results can be exploited in the scenarios tackled by OMNI.

To address the significant difficulties that delays within a process pose to PHD fitting, Korenčiak et al. [69] have tackled probabilistic regions of zero density by using interval distributions to separate discrete and continuous features of distributions. Similar work [70] supports the synthesis of timeouts in fixed-delay CTMCs by using Markov decision processes. Unlike OMNI, [70] and [69] do not consider essential non-Markovian features of real data such as multi-modal and long-tail distributions, and thus cannot handle empirical data that has these common characteristics.

Finally, non-PHD-based approaches to combining Markov models with real data range from using Monte Carlo simulation to analyse properties of discrete-time Markov chains with uncertain parameters [71] to using semi-Markov chains to model holding times governed by general distributions [72]. However, none of these approaches can offer the guarantees and tool support provided by OMNI thanks to its exploitation of established CTMC model checking techniques.

9 Conclusion

We presented OMNI, a tool-supported method for refining the CTMC models of component-based systems using observations of the component execution times. To evaluate OMNI, we carried out extensive experiments within two case studies from different domains. The experimental results show that OMNI-refined models support the analysis of transient QoS properties of component-based systems with greatly increased accuracy compared to the high-level CTMC models typically used in software performance engineering. Furthermore, we showed that significant accuracy improvements are achieved even for small training datasets of component observations, and for OMNI parameters corresponding to coarse-granularity refinements (and thus to relatively modest increases in model size).

In our future work, we plan to extend the applicability of OMNI to systems comprising components whose behaviour

changes during operation. This will require OMNI to continually refine the CTMC models of these systems based on new component observations. We envisage that this extension will enable the runtime analysis of QoS properties for rapidly evolving systems [73] and will support the dynamic selection of new configurations for self-adaptive software used in safety-critical and business-critical applications [74]. In addition, we intend to examine the effectiveness of OMNI in other application domains, and its ability to estimate a broader range of distributions for the execution times of the system components.

REFERENCES

[1] G. T. Leavens, M. Sitaraman, Foundations of component-based systems, Cambridge University Press, 2000.

[2] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, I. Meedeniya, Software architecture optimization methods: A systematic literature review, IEEE Transactions on Software Engineering 39 (5) (2013) 658–683.

[3] S. Becker, L. Grunske, R. Mirandola, S. Overhage, Performance prediction of component-based systems, in: R. H. Reussner, J. A. Stafford, C. A. Szyperski (Eds.), Architecting Systems with Trustworthy Components: International Seminar, Dagstuhl Castle, Germany, December 12-17, 2004. Revised Selected Papers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 169–192. doi:10.1007/11786160_10.

[4] H. Koziolek, Performance evaluation of component-based software systems: A survey, Performance Evaluation 67 (8) (2010) 634 – 658, special Issue on Software and Performance. doi:10.1016/j.peva.2009.07.007.

[5] A. Koziolek, D. Ardagna, R. Mirandola, Hybrid multi-attribute QoS optimization in component based software systems, Journal of Systems and Software 86 (10) (2013) 2542–2558.

[6] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, Modelling with generalized stochastic Petri nets, John Wiley & Sons, Inc., 1994.

[7] D. Perez-Palacin, et al., QoS and energy management with Petri nets: A self-adaptive framework, J. Syst. Softw. 85 (12) (2012) 2796–2811.

[8] G. Franks, T. Al-Omari, M. Woodside, O. Das, S. Derisavi, Enhanced modeling and solution of layered queueing networks, IEEE Transactions onSoftware Engineering 35 (2) (2009) 148–161.

[9] J. A. Carrasco, Computationally efficient and numerically stable reliability bounds for repairable fault-tolerant systems, IEEE Transactions on Computers 51 (3) (2002) 254–268.

[10] N. Sato, K. S. Trivedi, Stochastic modeling of composite web services for closed-form analysis of their performance and reliability bottlenecks, Springer, 2007.

[11] A. Hessel, K. G. Larsen, M. Mikucionis, B. Nielsen, P. Pettersson, A. Skou, Testing real-time systems using uppaal, in: R. M. Hierons, J. P. Bowen, M. Harman (Eds.), Formal Methods and Testing: An Outcome of the FORTEST Network, Revised Selected Papers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 77–117. doi:10.1007/978-3-540-78917-8_3.

[12] S. Becker, et al., The Palladio component model for model-driven performance prediction, J. Syst. Softw. 82 (1) (2009) 3–22.

[13] S. Baarir, M. Beccuti, D. Cerotti, M. De Pierro, S. Donatelli, G. Franceschinis, The GreatSPN tool: recent enhancements, ACM SIGMETRICS Performance Evaluation Review 36 (4) (2009) 4–9.

[14] M. Kwiatkowska, G. Norman, D. Parker, Prism 4.0: Verification of probabilistic real-time systems, in: Computer Aided Verification, Springer, 2011, pp. 585–591.

[15] G. Behrmann, A. David, K. G. Larsen, J. Hakansson, P. Petterson, W. Yi, M. Hendriks, Uppaal 4.0, in: Third International Conference on the Quantitative Evaluation of Systems - (QEST'06), 2006, pp. 125–126. doi:10.1109/QEST.2006.59.

[16] S. Gallotti, C. Ghezzi, R. Mirandola, G. Tamburrelli, Quality prediction of service compositions through probabilistic model checking, in: Quality of Software Architectures. Models and Architectures, Springer, 2008, pp. 119–134.

[17] S. Gerasimou, G. Tamburrelli, R. Calinescu, Search-based synthesis of probabilistic models for quality-of-service software engineering, in: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on, 2015, pp. 319–330. doi:10.1109/ASE.2015.22.

[18] A. Filieri, C. Ghezzi, G. Tamburrelli, A formal approach to adaptive software: continuous assurance of non-functional requirements, Formal Asp. Comput. 24 (2) (2012) 163–186.

[19] R. Calinescu, S. Gerasimou, A. Banks, Self-adaptive software with decentralised control loops, in: International Conference on Fundamental Approaches to Software Engineering, Springer, 2015, pp. 235–251.

[20] R. Calinescu, K. Johnson, Y. Rafiq, Developing self-verifying service-based systems, in: Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on, 2013, pp. 734–737. doi:10.1109/ASE.2013.6693145.

[21] P. Buchholz, J. Kriege, I. Felko, Phase-type distributions, in: Input Modeling with Phase-Type Distributions and Markov Models, Springer, 2014, pp. 5–28.

[22] A. Aziz, K. Sanwal, V. Singhal, R. Brayton, Verifying continuous time Markov chains, in: Computer Aided Verification, Springer, 1996, pp. 269–276.

[23] M. Z. Kwiatkowska, From software verification to 'everyware' verification, Computer Science - R&D 28 (4) (2013) 295–310. doi:10.1007/s00450-013-0249-1.

[24] C. Baier, E. M. Hahn, B. R. Haverkort, H. Hermanns, J. Katoen, Model checking for performability, Mathematical Structures in Computer Science 23 (4) (2013) 751–795. doi:10.1017/S0960129512000254.

[25] A. Filieri, G. Tamburrelli, C. Ghezzi, Supporting self-adaptation via quantitative verification and sensitivity analysis at run time, IEEE Trans. Software Eng. 42 (1) (2016) 75–99. doi:10.1109/TSE.2015.2421318.

[26] J. M. Franco, F. Correia, R. Barbosa, M. Z. Rela, B. R. Schmerl, D. Garlan, Improving self-adaptation planning through software architecture-based stochastic modeling, Journal of Systems and Software 115 (2016) 42–60. doi:10.1016/j.jss.2016.01.026.

[27] R. Calinescu, C. Ghezzi, K. Johnson, M. Pezzè, Y. Rafiq, G. Tamburrelli, Formal verification with confidence intervals to establish quality of service properties of software systems, IEEE Trans. Reliability 65 (1) (2016) 107–125. doi:10.1109/TR.2015.2452931.

[28] C. U. Smith, Software performance engineering, in: L. Donatiello, R. Nelson (Eds.), Performance Evaluation of Computer and Communication Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 1993, pp. 509–536.

[29] C. U. Smith, Introduction to software performance engineering: origins and outstanding problems, in: International School on Formal Methods for the Design of Computer, Communication and Software Systems, Springer, 2007, pp. 395–428.

[30] M. Woodside, G. Franks, D. C. Petriu, The future of software performance engineering, in: 2007 Future of Software Engineering, FOSE '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 171–187. doi:10.1109/FOSE.2007.32.

[31] C. Paterson, R. Calinescu, Accurate analysis of quality properties of software with observation-based Markov chain refinement, in: IEEE International Conference on Software Architecture, 2017, pp. 121–130.

[32] M. Kwiatkowska, G. Norman, D. Parker, Stochastic model checking, in: Intl. Conf. on Formal Methods for Performance Eval., 2007, pp. 220–270.

[33] G. Norman, D. Parker, Quantitative verification: Formal guarantees for timeliness, reliability and performance, Tech. rep., London Mathematical Society and the Smith Institute for Industrial Mathematics and System Engineering (2014).

[34] S. Gerasimou, R. Calinescu, G. Tamburrelli, Synthesis of probabilistic models for quality-of-service software engineering, Automated Software Engineering Journal (2018), Springer. doi:10.1007/s10515-018-0235-8.

[35] R. Calinescu, M. Autili, J. Cámara, A. Di Marco, S. Gerasimou, P. Inverardi, A. Perucci, N. Jansen, J.-P. Katoen, M. Kwiatkowska, O. J. Mengshoel, R. Spalazzese, M. Tivoli, Synthesis and verification of self-aware computing systems, in: S. Kounev, J. O. Kephart, A. Milenkoski, X. Zhu (Eds.), Self-Aware Computing Systems, Springer, 2017, pp. 337–373. doi:10.1007/978-3-319-47474-8_11.

[36] R. Calinescu, M. Češka, S. Gerasimou, M. Kwiatkowska, N. Paoletti, Efficient synthesis of robust models for stochastic systems, Journal of Systems and Software (2018). doi:10.1016/j.jss.2018.05.013.

[37] J. R. Norris, Markov chains, Cambridge University Press, 1998.

[38] C. Baier, B. Haverkort, H. Hermanns, J. P. Katoen, Model-checking algorithms for continuous-time Markov chains, IEEE Transactions on Software Engineering 29 (6) (2003) 524–541. doi:10.1109/TSE.2003.1205180.

[39] W. J. Anderson, Continuous-time Markov chains: An applications-oriented approach, Springer Science & Business Media, 2012.

[40] G. Yin, Q. Zhang, Continuous-time Markov chains and applications: a two-time-scale approach, Vol. 37, Springer Science & Business Media, 2012.

[41] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, D. N. Jansen, The ins and outs of the probabilistic model checker MRMC, Performance evaluation 68 (2) (2011) 90–104.

[42] C. Dehnert, S. Junges, J. Katoen, M. Volk, A Storm is coming: A modern probabilistic model checker, in: R. Majumdar, V. Kuncak (Eds.), 29th International Conference on Computer Aided Verification (CAV), Vol. 10427 of Lecture Notes in Computer Science, Springer, 2017, pp. 592–600. doi:10.1007/978-3-319-63390-9_31.

[43] M. Fackrell, Modelling healthcare systems with phase-type distributions, Health care management science 12 (1) (2009) 11–26.

[44] E. Ishay, Fitting phase-type distributions to data from a telephone call center, Ph.D. thesis, Technion-Israel Institute Of Technology (2002).

[45] A. H. Marshall, M. Zenga, Simulating Coxian phase-type distributions for patient survival, International Transactions in Operational Research 16 (2) (2009) 213–226.

[46] C. A. O'Cinneide, Phase-type distributions: open problems and a few properties, Communications in Statistics. Stochastic Models 15 (4) (1999) 731–757.

[47] W. J. Stewart, Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling, Princeton University Press, 2009.

[48] C. E. da Silva, J. D. S. da Silva, C. Paterson, R. Calinescu, Self-adaptive role-based access control for business processes, in: Proceedings of the 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, IEEE Press, 2017, pp. 193–203.

[49] R. Calinescu, Y. Rafiq, Using intelligent proxies to develop self-adaptive service-based systems, in: 7th International Symposium on Theoretical Aspects of Software Engineering, 2013, pp. 131–134. doi:10.1109/TASE.2013.41.

[50] C. Ghezzi, M. Pezzè, M. Sama, G. Tamburrelli, Mining behavior models from user-intensive web applications, in: Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, pp. 277–287.

[51] G. Su, D. S. Rosenblum, Asymptotic bounds for quantitative verification of perturbed probabilistic systems, in: International Conference on Formal Engineering Methods, Springer, 2013, pp. 297–312.

[52] A. Bobbio, A. Cumani, Ml estimation of the parameters of a ph distribution in triangular canonical form, Computer performance evaluation 22 (1992) 33–46.

[53] P. Reinecke, T. Krauß, K. Wolter, Cluster-based fitting of phase-type distributions to empirical data, Computers & Mathematics with Applications 64 (12) (2012) 3840–3851.

[54] A. Horvath, M. Telek, Approximating heavy tailed behaviour with phase type distributions, http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.6351 (2000).

[55] H. Okamura, T. Dohi, Fitting phase-type distributions and Markovian Arrival Processes: Algorithms and tools, in: Principles of Performance and Reliability Modeling and Evaluation, Springer, 2016, pp. 49–75.

[56] A. Bobbio, A. Horváth, M. Telek, Matching three moments with minimal acyclic phase type distributions, Stochastic Models 21 (2–3) (2005) 303–326.

[57] A. Horváth, M. Telek, Matching more than three moments with acyclic phase type distributions, Stochastic Models 23 (2) (2007) 167–194. doi:10.1080/15326340701300712.

[58] S. Asmussen, O. Nerman, M. Olsson, Fitting phase-type distributions via the EM algorithm, Scandinavian Journal of Statistics (1996) 419–441.

[59] A. Thummler, P. Buchholz, M. Telek, A novel approach for phase-type fitting with the EM algorithm, IEEE Transactions on Dependable and Secure Computing 3 (3) (2006) 245–258. doi:10.1109/TDSC.2006.27.

[60] J. Wang, J. Liu, C. She, Segment-based adaptive hyper-Erlang model for long-tailed network traffic approximation, The Journal of Supercomputing 45 (3) (2008) 296–312. doi:10.1007/s11227-008-0173-5.

[61] H. Okamura, R. Watanabe, T. Dohi, Variational bayes for phase-type distribution, Communications in Statistics - Simulation and Computation 43 (8) (2014) 2031–2044.

[62] Y. Yamaguchi, H. Okamura, T. Dohi, A variational Bayesian approach for estimating parameters of a mixture of Erlang distribution, Communications in Statistics - Theory and Methods 39 (13) (2010) 2333–2350.

[63] P. Reinecke, T. Krauss, K. Wolter, Hyperstar: Phase-type fitting made easy, in: 2012 Ninth International Conference on Quantitative Evaluation of Systems, IEEE, 2012, pp. 201–202.

[64] P. Reinecke, T. Krauß, K. Wolter, Phase-type fitting using Hyperstar, in: Computer Performance Engineering, Springer, 2013, pp. 164–175.

[65] S. A. Anichkin, Hyper-Erlang approximation of probability distributions on $(0, \infty)$ and its application, in: V. V. Kalashnikov, V. M. Zolotarev (Eds.), Stability Problems for Stochastic Models, 1983, pp. 1–16.

[66] D. N. Jansen, J.-P. Katoen, M. Oldenkamp, M. Stoelinga, I. Zapreev, How fast and fat is your probabilistic model checker? An experimental performance comparison, in: K. Yorav (Ed.), Hardware and Software: Verification and Testing: Third International Haifa Verification Conference, HVC 2007, Springe, 2008, pp. 69–85.

[67] P. Karmakar, K. Gopinath, Are Markov models effective for storage reliability modelling?, arXiv preprint arXiv:1503.07931.

[68] G. Ciobanu, A. S. Rotaru, Phase: a stochastic formalism for phase-type distributions, in: Formal Methods and Software Engineering, Springer, 2014, pp. 91–106.

[69] L. Korenčiak, J. Krčál, V. Řehák, Dealing with zero density using piecewise phase-type approximation, in: Computer Performance Eng., Springer, 2014, pp. 119–134.

[70] T. Brázdil, L. Korenčiak, J. Krčál, P. Novotnỳ, V. Řehák, Optimizing performance of continuous-time stochastic systems using timeout synthesis, in: Quantitative Evaluation of Systems, Springer, 2015, pp. 141–159.

[71] I. Meedeniya, I. Moser, A. Aleti, L. Grunske, Evaluating probabilistic models with uncertain model parameters, Software & Systems Modeling 13 (4) (2014) 1395–1415.

[72] G. G. I. López, H. Hermanns, J.-P. Katoen, Beyond memoryless distributions: Model checking semi-Markov chains, in: L. de Alfaro, S. Gilmore (Eds.), Process Algebra and Probabilistic Methods, Springer Berlin Heidelberg, 2001, pp. 57–70. doi:10.1007/3-540-44804-7_4.

[73] H. A. Müller, N. M. Villegas, Runtime evolution of highly dynamic software, in: T. Mens, A. Serebrenik, A. Cleve (Eds.), Evolving Software Systems, Springer, 2014, pp. 229–264. doi:10.1007/978-3-642-45398-4_8.

[74] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, T. Kelly, Engineering trustworthy self-adaptive software with dynamic assurance cases, IEEE Transactions on Software Engineering PP (99) (2017) 1–31. doi:10.1109/TSE.2017.2738640.

Colin Paterson is a Research Associate in the Assuring Autonomy International Programme at the University of York, where his research considers techniques for the verification of artificial intelligence. Colin in currently completing a PhD which concerns the formal verification of operational processes using observation data to enhance the modelling of such processes and the accuracy of verification techniques.

Prior to this Colin obtained a PhD in control systems engineering in a collaboration with Jaguar Cars, before moving into industry where he designed bespoke web-based software solutions as well as a product suite for local government focused on governance, risk and compliance.

Radu Calinescu is a Senior Lecturer within the Department of Computer Science at the University of York, UK. His main research interests are in formal methods for self-adaptive, autonomous, secure and dependable software and artificial intelligence systems, and in performance and reliability software engineering. He is an active promoter of formal methods at runtime as a way to improve the integrity and predictability of self-adaptive and autonomous software systems and processes.